BYJU'S EXAM PREP

Computer Science & IT

Operating Systems

SHORT NOTES

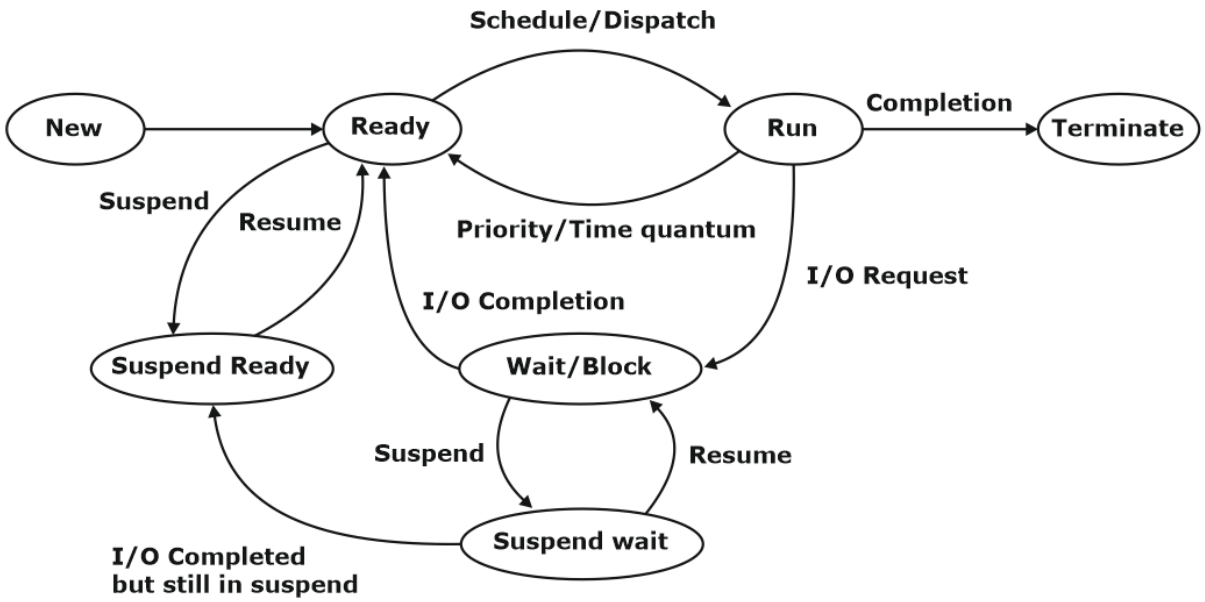
Short Notes — OPERATING SYSTEMS

- **Operating System**: - An interface between user and system hardware.
- **Types of operating systems: -**
  - Batch operating system
  - Time-sharing operating system
  - Distributed operating system
  - Network Operating system
  - Real-time operating system
- **System Call**- It is a request made by a user program to the operating system to perform any task.
- **Program**- It is a set of instructions, and by executing them step by step we can get any service done, it is a passive entity and resides on the disk.
- **Process**- Instance of program or a program under execution. It is an active entity which resides in the main memory.

  Operations on Process: -
  - Creating process
  - Running, scheduling, Suspending, Resuming, Blocking
  - Terminating a process
- **Modes of operating system**

  Operating system operates in two modes, which are: -
  - User Mode
    - It is also known as non-privileged mode.
    - Preemption is always possible
    - All user programs execute in this mode.
  - Kernel Mode
    - It is also known as privileged mode, supervisory mode, system mode.
    - No preemption is allowed.
    - Operating system's program run in this mode.
- **Process Control Block-** Process Control Block is defined as a data structure that contains all the information related to a process. The process control block is also called task control block or entry of the process table, etc.

- **Attributes of PCB: -**
  - Process ID
  - Program Counter status
  - Process state
  - Memory Management data
  - CPU scheduling data
  - CPU registers
  - Address space for process
  - Accounting data
  - I/O information
  - User's Identification
  - List of open files
  - List of open devices
  - Priority
- **Process State Diagram:**



Process State Diagram

- **Threads-** A thread may be defined as a flow of execution through some process code.
  - A thread is also known as a **lightweight process**.
  - Threads improve application's performance by providing parallelism.
  - Threads improve the performance of operating systems by reducing the process overhead.
  - Thread is equivalent to a classical process and it is a software approach for software improvement.
  - Every thread belongs to only one process and not a single thread can exist outside a process.

- **Types of Threads-**
  - There are two types of threads, which are:
    - User Level Threads
    - Kernel Level Threads
- **Difference between ULT and KLT**

USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread performs blocking operation, then another thread can continue execution.
User level threads are designed as dependent threads.	Kernel level threads are designed as independent threads.

- Why do we need Threads over Process?
  - Creating a process and switching between processes is costly.
  - Processes have separate address space
  - Sharing memory areas among processes is non-trivial.

- **Difference between Process and threads:**

Process	Thread
Process is resource intensive or heavy weight.	Thread is a light-weight process which requires less resources than a process.
Process switching needs interaction with the OS.	Thread switching doesn't require any interaction with the operating system.

Process	Thread
In multiprocessing environments, each process has its own memory and file resources but executes the same code.	All threads can share the same open files, child processes.
If one process gets blocked, then another process cannot run until the first process is unblocked.	Whereas if one thread gets blocked and waiting, a second thread of the same task can execute.
Multiple processes without threads needs more resources.	Multi-threaded processes use less resources.
In multiple processes each process operates independently of the others.	Thread can read, write or alternate another thread's data.

- **Multithreading-** It is an ability of the CPU to execute multiple threads concurrently.
  - Advantages of Multithreading:
    - Resource sharing
    - Responsiveness
    - Less context switching overhead i.e., better economical design.
    - Efficiently utilize multiprocessor architecture.
  - Multithreading Models:
    - Many-to-one model
    - One-to-one model
    - Many-to-model
- **Schedulers in OS**- Schedulers in OS are special system software. They help in scheduling the processes in various ways.

  Types of Schedulers:
  - Long-term Scheduler
    - A long-term scheduler is to maintain a good degree of multiprogramming.
    - Long-term scheduler is also known as **Job Scheduler**.
  - Short-term Scheduler
    - A short-term scheduler is to increase the system performance.
    - Short-term scheduler is also known as **CPU Scheduler**
  - Medium-term Scheduler
    - A medium-term scheduler is to perform swapping
    - The medium-term scheduler reduces the degree of multiprogramming.
- **Dispatcher-** The dispatcher is the module that gives control of the CPU to the process selected by the scheduler. This function involves:

- Switching context.
- Switching to user mode.
- Jumping to the proper location in the newly loaded program.

The dispatcher needs to be as fast as possible, as it is run on every context switch. The time consumed by the dispatcher is known as dispatch latency.

- **Process Queues**- The Operating system manages various types of queues for each of the process states. The PCB related to the process is also stored in the queue of the same state. If the Process is moved from one state to another state, then its PCB is also unlinked from the corresponding queue and added to the other state queue in which the transition is made.
  - Queues Maintained by Operating System:
    - Job queue
    - Ready queue
    - Waiting queue
- Scheduling Criteria-
  - **CPU Utilization:** CPU should be as busy as possible.
  - **Throughput:** Number of processes completed per unit.
    Throughput= N/L,
    where N = number of processes,
    and L = Maximum Completion Time – Minimum Arrival Time
  - **Arrival Time:** Time at which the process arrives in the ready queue.
  - **Completion Time:** Time at which process completes its execution.
  - **Burst Time:** Time required by a process for CPU execution.
  - **Turn Around Time:** Completion Time – Arrival Time
  - **Waiting Time (W.T):** Turnaround Time – Burst Time
- **Starvation**- Indefinite waiting of processes to get CPU cycles for execution.
- **CPU Scheduling Algorithm**
  - **FCFS (First Come First Serve)-**
    - It schedules the processes according to the order of their arrival time.
    - Non-preemptive in nature.
    - No starvation.
    - Simple
    - Easy to understand and implement
    - In this if shorter jobs arrive later than long jobs, so shorter jobs have to wait for a long time, and this phenomenon is known as **Convoy Effect.**
  - **Shortest Job First(SJF) or Shortest Job Next (SJN)-**
    - It executes jobs with smaller CPU burst first.
    - Non-preemptive in nature.
    - Gives less average waiting time than all other scheduling algorithms
    - Jobs with large burst time may starve the CPU.

    - Poor response time
    - Unimplementable, because it is impossible to predict burst time of processes in advance.
    - Used as Benchmark to measure performance of all other processes.
    - High Throughput.
  - **Shortest Remaining Job first (SRTF)-**
    - It first executes jobs who have the smallest remaining burst time.
    - Pre-emptive in nature.
    - Jobs with large burst time may starve the CPU.
    - Poor response time, but better than SJF.
    - High throughput.
    - Unimplementable, because it is impossible to predict burst time of processes in advance.
- Round Robin
  - Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called *time quantum*.
  - Always Pre-emptive in nature.
  - Best response time.
  - Ready queue is treated as a circular queue.
  - Mostly used for a time-sharing system.
  - With large time quantum, it acts as FIFO
  - No priority.
  - No starvation.
- Priority Scheduling
  - Every process assigned with a number, and these numbers will be the priority of that process.
  - Processes with lower priority may starve for CPU.
  - No idea of response time
  - Mostly used in Real-time operating systems.
  - Can be preemptive and non-preemptive both in nature.
- Longest Job First
  - First executes the process with longer CPU burst.
  - Completely opposite of SJF.
  - Non-preemptive in nature.
  - Starvation exists.
- Longest Remaining Time First
  - First executes the process with longer CPU burst.
  - Completely opposite of SRTF.
  - Preemptive in nature.
  - Starvation exists.
- Highest Response Ratio Next

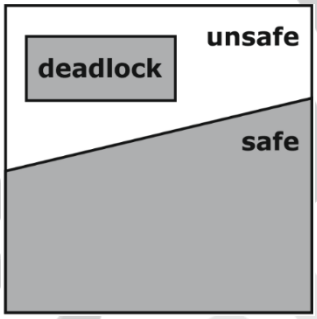
- A Response Ratio is calculated for each of the available jobs and the Job with the highest response ratio is given priority over the others.
- Response Time= (W+S)/S Where, W → Waiting Time S → Service Time or Burst Time
- Most optimal scheduling algorithm.
- Non-preemptive in nature.
- No starvation.

- Multilevel Queue Scheduling
  - As only one ready queue is there so only one scheduling technique is used. Therefore, to use more than one scheduling can be used with multilevel queue scheduling.
  - Each queue can have different priorities.
  - Starvation exists.
- Multilevel Feedback Queues
  - Similar to Multilevel queue scheduling, except jobs may be moved from one queue to another.
  - It can adjust each job's priority.
  - Improved version of Multilevel queue scheduling.
  - No starvation.
- **Deadlock**- A Deadlock is a situation where each of the computer processes waits for a resource which is being assigned to some other process. In this situation, none of the process gets executed since the resource it needs is held by some other process which is also waiting for some other resource to be released.
  Minimum two processes are required for deadlock to occur.
- **Necessary Conditions of Deadlock:**
  - **Mutual exclusion**- Mutual exclusion simply means that two resources at the same time cannot access a single resource.
  - **Hold and Wait-** Hold and Wait says that a process is waiting for some resources while keeping another resource with it at the same time.
  - **No-preemption**- A process holding a resource will release it voluntarily after the completion of the process, and no other process can take that resource from the process holding it.
  - **Circular Wait**- A process A is holding resource R3 and waiting for resource R1 and another process B holding resource R1 but waiting for some other resource R2 and resource R2 is occupied by some process C which is waiting for resource R3. In this situation all the processes are waiting for each other in a cyclic manner, no process wants to release the resource and continues to wait for others. In this way all the processes wait for each other for indefinite time and can lead to deadlock.
- Strategies For Handling Deadlock
  - **Deadlock Prevention:** Don't let a deadlock occur in the system by violating any one of the necessary conditions. It involves:
    - Infinite resources

- No sharing
- No waiting
- Preempt resources
- Allocate all resources at the beginning.
- Every process use the same order for accessing resource

  - To violate any of the four conditions
    - Mutual exclusion: Cannot be violated as this condition is hardware dependent
    - Hold & Wait: Conservative approach; Do not hold; Wait timeout.
    - Non-Preemption: To violate forcefully preemption.
    - Circular-wait: Give unique numbers to each process either in increasing or decreasing order. If a process requires less no. of resource, it must release all higher no. of resource & request for all required no. again.

- **Deadlock Avoidance:** Avoid occurrence of deadlock in the system by various methods. Checks whether the system is in a safe state or not.



  - A system is said to be safe if and only if the needs of all the processes could be satisfied with the available resources in some order. The order is known as a safe sequence.
  - There may exist multiple safe sequences.
  - Single instance of resources using resource allocation graph.
  - Banker's Algorithm or Safety Algorithm is used to allocate multiple instances of resources to the processes.
  - Need= Max- Allocation,
  - r>= p (n-1) +1; where, r = number of available resources (deadlock free), n= maximum number of resources needed by each process, p = number of processes.

- **Deadlock detection and Recovery:** Do not check for any safety and whenever any process requests some resources then allocate those resources immediately. If a deadlock occurs then identify it and recover it by suitable methods.
  - Deadlock detection involves:
    - Scan the RAG (Resource allocation Graph)
    - Detect a cycle in a graph.
    - Recovery from deadlock.
    - When we activated the detection algorithm, CPU utilization has decreased or no CPU utilization. Most of the processes are blocked.

- Deadlock recovery involves:
  - Abort all deadlocked processes.
  - Abort one process at a time and check, repeat this process until deadlock is removed.
  - Rollback to safe state and restart from that state.
  - Preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

- **Deadlock Ignorance:** This involves completely ignoring the concept of deadlock, according to this no deadlock exists. It uses the Ostrich approach.
- **Inter-process Communication-**
  - Independent processes- Two or more processes will execute autonomously without affecting the output or order of execution of other processes.
    - No states will be shared with other processes
    - Order of scheduling doesn't matter.
    - Output of a process is independent of order of execution of other processes.
  - Co-operating processes- Two or more processes are said to be co-operative if and only if they get affected by or affect the execution of other processes.
    - These processes require an IPC mechanism that will allow them to exchange information and data like messages, shared memory, shared variables, etc.
    - Shared resources, speed-up and modularity are the advantages of co-operating processes.
- IPC Models- There are primarily two IPC models:
  - Shared memory
  - Message Passing
- **Concurrency**
  - Able to run multiple applications at the same time.
  - Allows better resource utilization
  - Better performance
  - Better average response time
- **Synchronization**- The procedure which is involved in preserving the order of execution of cooperative processes is called Process Synchronization.

  Need of Synchronization-
  - When two or more processes execute concurrently while sharing some system resources. So, all these processes execute properly, we need proper synchronization between their actions.
  - Process synchronization is to avoid the inconsistent output.
- Critical Section- Critical section is a small piece of code or a section in the program from where a process accesses the shared resources concurrently during its execution.
- Race Condition- The final output produced completely depends upon the execution order of instructions of processes.

- Criteria of Synchronization Mechanism-
  - **Mutual Exclusion-** It says that two processes cannot be present in the critical section simultaneously.
  - **Progress-** Process running outside the critical section can never block other interested processes from entering the critical section when it is free.
  - **Bounded waiting-** Due to mutual exclusion the waiting time of a process must be finite.
- Lack of Process Synchronization can generate-
  - Inconsistency in the system.
  - Loss of data
  - Deadlock
- **Semaphore-** A semaphore is a integer variable that provide synchronization between multiple processes executing concurrently in a system.
- **Operations on Semaphore**
  - Wait operation
    - Also known as down, or P operation.
    - It decreases semaphore's value by 1.
    - It is used in the entry part of the critical section.
    - If a resource is not free, then block the process.
  - Signal operation
    - Also known as UP, or V operation.
    - It increases semaphore's value by 1.
    - It is used in the exit part of the critical section to release the acquired lock.
- **Types of semaphore-**
  - Binary semaphore- Uses limited number, only 0 and 1.
    - It is busy waiting, known as the spin lock issue.
    - It can work on a single instance of a resource.
    - It's wait operation should complete only for a single process.
  - Counting Semaphore- It can use any integer.
    - Its negative value indicates the number of requests in a waiting queue.
    - Its positive value indicates the number of instances of resource R currently available for allocation, and there is no request in the waiting queue.
    - While using multiple semaphores, following proper ordering is highly important.
  - Strong semaphore- It is that semaphore whose definition includes policy of FIFO queue.
  - Weak semaphore- A semaphore that doesn't specify the order in which processes are removed from the queue.
- **Classical Synchronization Problems**
  - Sleeping barber problem
  - Bounded buffer problem

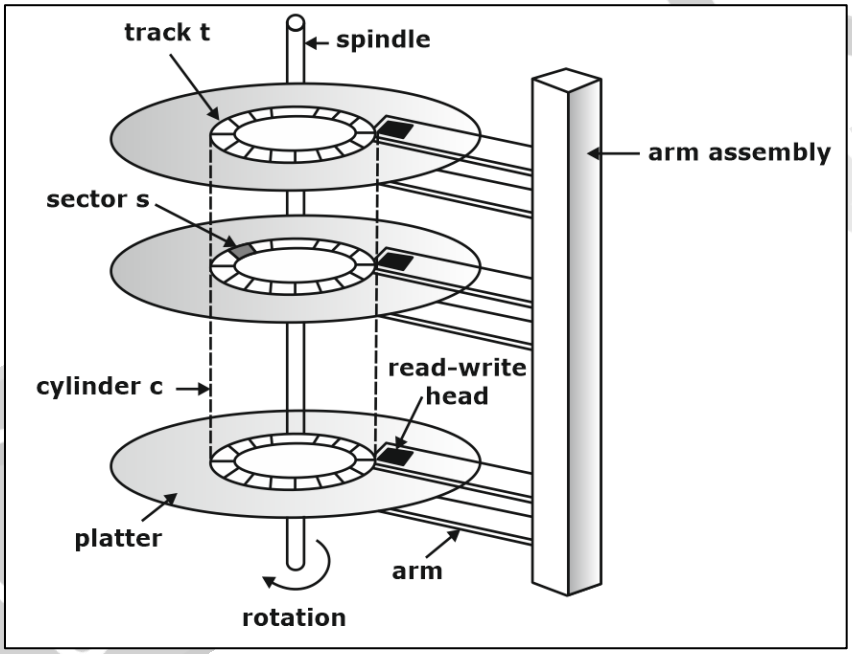
- Producer Consumer problem
- Dining Philosophers Problem
- Reader's writer's Problem

- **Synchronization Mechanism**
  - Lock variable
    - Software mechanism implemented in user mode.
    - Solution to busy waiting.
    - Completely failed mechanism because it cannot satisfy basic criteria of synchronization, i.e., mutual exclusion.
  - Test and Set Lock
    - Uses test & Set instruction to maintain synchronization between processes executing concurrently.
    - Ensures mutual exclusion.
    - Deadlock free.
    - Does not guarantee bounded waiting.
    - Some processes may suffer from starvation.
    - Suffers from spin lock.
    - It is not neutral architecturally because it needs the OS to support test-and-set instruction.
    - It is a solution to busy waiting.
  - Turn variable
    - It uses a variable called turn to provide the synchronization among processes.
    - Ensures mutual exclusion.
    - Follows a strict alternation approach, so progress is not guaranteed.
    - Ensures bounded waiting because processes execute one by one and there is a guarantee that each process will get a chance to execute.
    - Starvation doesn't exist.
    - It is neutral architecturally because it doesn't need any support of the operating system.
    - Deadlock free.
    - It is a solution to busy waiting.
  - Strict Alternation Approach
    - Processes have to enter the critical section compulsorily in an alternate order whether they want to enter or not.
    - Ensures mutual exclusion.
    - Doesn't follow the strict alternation approach.
    - Ensures progress.
    - It is neutral architecturally.
    - It is a solution to busy waiting.
    - Suffers from **deadlock**.

- No guarantee of bounded waiting.

- **Peterson Solution**
  - It is a synchronization mechanism implemented in user mode, it uses two variables: turn and interest.
  - Satisfy mutual exclusion.
  - Satisfy progress.
  - Bounded waiting is not guaranteed.

- **Spin Lock and Busy Waiting-** Spinlock is a lock that causes a thread trying to acquire it to simply wait in a loop while repeatedly checking if the lock is available. Since the thread remains active but it is not performing a useful task, the use of such a lock is kind of Busy waiting.
- **Disk -** In computers the secondary storage space is typically formed of stacked up magnetic disks one on the top of another. One such single unit of disk is known as a platter.



Data -> Sector -> Track -> Surface -> Platter -> Disk

- **Transfer rate:** Transfer rate is defined as the rate at which data is transferred from disk to the computer.
  Transfer rate = Number of heads × Capacity of one track × Number of rotations per second
- **Random access time:** Random access time is the sum of the seek time and rotational latency.
- **Seek Time**: The time taken by the arm to locate the desired track.
- **Rotational Latency:** The time taken by the arm to locate the required sector in the track.
- **Transfer time:** Time taken to transfer data.
- **Queue time:** In case, if a buffer is full of requests & more requests are coming, so we send coming requests to a queue, and the time requests wait in the queue know as Queue time.

- **Disk Access Time =** Seek time + Rotational Latency + Transfer time + Controller time + Queue time

- **Disk Scheduling-** Disk scheduling is a mechanism used by the operating system to schedule multiple disk access requests.
- **Various Disk Scheduling Algorithms**
  - FCFS (First Come First Serve)- FCFS first come first serve disk scheduling algorithm, as its name suggests this algorithm serves the disk requests who come first in the system.
    - Simple
    - Easy to implement and understand.
    - No starvation
    - Increased totals seek time.
    - Inefficient algorithm
  - SSTF (Shortest Seek Time First)- It is based on the concept that we will service that request next which requires the least number of head movements from the current position of the head regardless of the direction.
    - Less seek time
    - Good average response time
    - Complex than FCFS.
    - Starvation exists.
    - High variance in waiting time and response time
    - Frequently switching head's direction will slow down the algorithm.
- **SCAN algorithm-** It scans all the cylinders in the disk block back and forth.
  - Also known as the elevator algorithm.
  - No starvation.
  - Better than FCFS.
  - Complex to implement
  - Easy to understand
  - Gives low variance in waiting time and response time.
  - The head has to move till the end of the disk even if there are no disk requests to be serviced. This leads to unnecessary head movement and results in increased total seek time.
- LOOK algorithm- LOOK Algorithm is another improved version of the **SCAN Disk Scheduling Algorithm**. This algorithm is based on the concept that the head pointer starts from the first disk request at one end and moves towards the last request at the other end of the disk servicing all the requests in between.
  - No starvation
  - Better performance than SCAN algorithm.
  - Gives low variance in waiting time and response time.
  - Cylinders just visited by the head must wait for a long time.

- C-SCAN (Circular- SCAN) algorithm- Circular-SCAN algorithm is an improved version of the **SCAN Disk Scheduling Algorithm**. This algorithm is based on the concept that the head starts from one end and moves towards the other end of the disk while servicing all the requests in between. Once the head reaches the other end of the disk, its direction reverses.
  - Better response time
  - Uniform waiting time
  - More seek movements as compared to the SCAN algorithm
- C-LOOK (Circular-LOOK) algorithm- C-LOOK algorithm based on the concept that the head starts from the first request at one end and moves towards the last request at the other end of the disk while servicing all the requests in between. After reaching the last request at the other end, the head reverses its direction. Then the head pointer returns to the first request at the starting end of the disk without servicing any request in between. The same process continues until all the processes are serviced.
  - Reduces waiting time.
  - Have better performance than LOOK Algorithm.
  - No starvation exists.
  - It provides low variance in waiting time and response time.
  - There is an extra overhead of determining the end requests.
- **Memory Management**
  - Dynamic Loading- It refers to loading the library into the memory during load or run time. All routines are kept in "relocatable format" and a routine is not loaded until it isn't called.
  - Dynamic Linking- It refers to the linking that is done during load and runtime and not when the **.exe** file is created. Dynamic linking needs the operating system's support to perform its operation. Library which links dynamically is known as Dynamic Link Library (DLL).
  - Static Linking- All modules of the program are stored in memory before they are called.
- **Memory Management Techniques**

  We have two primary techniques for memory management, which are:
  - Contiguous Memory Allocation -
    - Program is stored at one place only- Centralized approach
    - Multiprogramming with fixed tasks (MFT).
    - Multiprogramming with variable tasks (MVT).
    - Overlays
    - Buddy System
  - Non-Contiguous Memory Allocation-
    - Programs are not centralized; it is distributed in memory according to the availability.
    - Paging
    - Segmentation
    - Segmented paging
    - Virtual memory (Demand paging)

- **Partitions-** Memory of a computer system is divided into some small sections, so that the program can use them.
  - Functions can be used for partitions:
    - Allocation
    - Deallocation
    - Protection (by using limit registers)
    - Free space management
- **Goals of Partition**
  - Minimum memory wastage (fragmentation).
  - Executing larger programs in smaller memory area

- **Contiguous Memory Allocation**- we have two ways which support the above goals of partitions, those ways are overlays, and virtual memory. This will increase the degree of multiprogramming which will eventually increase the CPU utilization and throughput.
- Overlays-
  - It is a very old memory management technique that focus on a single process information retained in RAM at a time.
  - It is best suited for sequential processing, but not for multitasking and multiprogramming environments.
  - Overlaying is possible when the programs are divided into modules and each module is independent.
- Multiprogramming with Fixed Task
  - Maximum process size is proportional to the maximum partition size
  - Degree of multiprogramming is proportional to number of partitions
  - This suffers from Internal as well as external fragmentation.
- Multiprogramming with Variable Task
  - Degree of multiprogramming varies.
  - It suffers from external fragmentation.
  - Worst fit is the best allocation policy for this.
- Buddy System-
  - The 'buddy system' allocates memory from a fixed size segment consisting of physically contiguous pages.
  - Memory is allocated from this segment using a power-of-2-allocator, which satisfies the requests in units sized as a power of 2.
  - Advantage of the buddy system is that how fast adjacent buddies can be combined to form larger segments using a technique called coalescing.
- Allocation Policies
  - First Fit- Allocate first available free space.

- Best Fit- Allocate best available space.
- Worst Fit- Allocate largest available space.
- Next Fit- Similar to first fit, but it starts from last allocated processes.

- Internal Fragmentation- In fixed size partitioning if process size is smaller than partition size unused free space is left within partition, and these are known holes/ memory fragments/ unused free space/ waste memory and their sum is known as Internal fragmentation.
- External fragmentation- In variable size partitioning initially processes are contiguous but as processes leave their space these partitions may not suit the new processes for allocation. Those unused partitions are known as External fragmentation.
  - Solutions of external fragmentation: -
    - Compaction
    - Non-contiguous memory allocation
- Compaction: The phenomenon when free memory chunks are collected together and it results in larger memory chunks which can be used as space available for processes, this mechanism is referred to as Compaction. In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out. Compaction is simply a mechanism of combining all the empty spaces together to form large free space so that we can use that free space for processes.
  - It increases CPU overhead.
  - Program needs dynamic allocation.
- **Non-Contiguous Allocation**- Size of Logical Address Space (LAS) generated by CPU is greater than or equal to size of physical address space (PAS) generated by Main memory. OS that there will be efficient use of multiprogramming, can run larger programs in smaller memory areas.
- **Paging:** It is a memory management technique that fulfills lack of contiguous memory allocation of physical memory.
- **Physical Address Space**
  - Physical address space is divided into equal frame size.
  - Page size= Frame size
  - Number of frames (N) = physical address space/ page size or frame size
  - Bits used for Frame number = $\lceil \log_2 N \rceil$
  - Frame offset = Page offset = d
- **Logical Address Space**
  - Logical address space is the size of the process. The size of the process should be less enough so that it can reside in the main memory.
  - Logical Address space is divided into equal size pages.
  - Page size is generalization of power of 2.
  - Number of pages in LAS (M) = Logical address space/ page size
  - Bits required for page number (P)= $\lceil \log_2 M \rceil$

- Page offset depends on number of page size
- Page number dependents on number of pages.

- **Page table-**
  - Page Table may be defined as a data structure used to store the mapping between logical addresses and physical addresses in the virtual memory system.
  - Logical addresses are produced by the CPU for the page of the processes that's why logical addresses are used by the processes generally.
  - Physical addresses are the actual frame address which is present in the main memory. Hardware or more specifically by RAM subsystems make use of physical addresses generally.
  - Main memory stores the Page table.
  - Number of pages of a process is equal to the number of Page table entries.
  - Page Table Base Register (PTBR) stores the page table's base address.
  - There is an independent page table for each process.

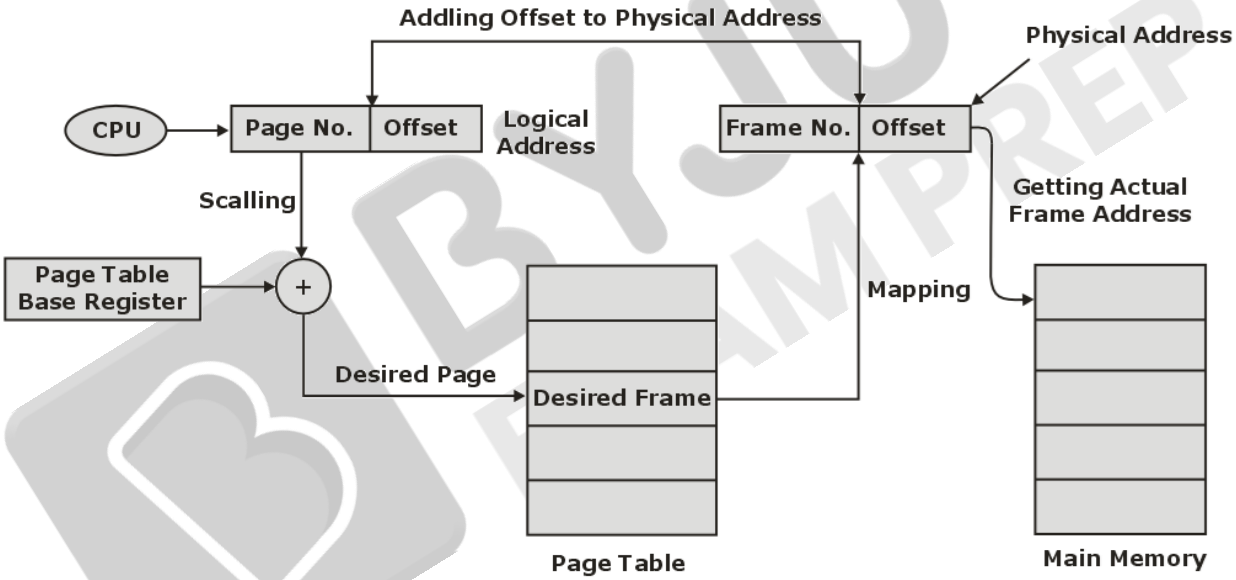
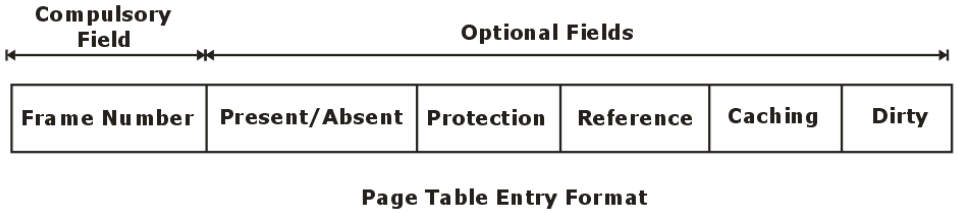


Figure - Mapping from page table to main memory

- Number of entries in a page table = Number of pages in logical address space
- Page Table entry contains frame number.

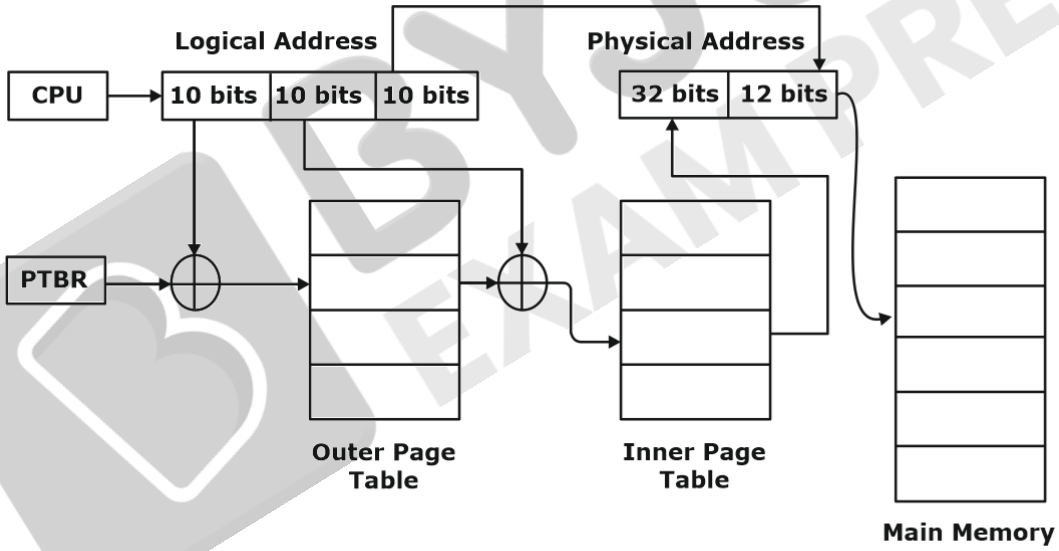


Page Table Entry Format

- Generally, Each entry of a page table have the following information
  - Frame number describes the frame where the page is actually stored in the main memory.

- Present/ Absent (valid/ invalid) bit is used to specify whether that page is present in the primary memory or not.
- Protection (read/write) bit specifies the permission to perform r/w operation on a page.
- Reference bit specifies whether the page has been referenced in the latest clock cycle or not.
- Caching is used to enable or disable the caching of a page.
- Dirty bit is used to specify whether that page is modified or not.

- **Types of Page Table-**
  - Single level Page table- This is the most simple and straightforward approach that has a single linear array of page-table entries (PTEs). Each PTE stores information about the page, such as its physical page number ("frame" number) as well as status bits, such as whether the page is valid.
  - Multi-level Page table- multi-level page tables are tree-like structures to hold page tables. It is a paging scheme where a hierarchy of page tables exists. The entries of the level-0-page table are pointers to a level-1 page table, and the entries of the level-1 page table are PTEs as described above in the single-level page table.



Multi-Level Paging

- **Drawbacks of Paging-**
  - For a larger process the size of the Page table will be very large, and it will waste main memory.
  - While reading a single word from the main memory CPU will take more time.
  - How to decrease the page table size- The page table size can be reduced by increasing the page size, but this will cause internal fragmentation and page wastage too.
  - Another method is to use multilevel paging, but this will increase the effective access time, so this is not a practical method.

- How to decrease the effective access time- CPU can use a register with a page table stored in it so that the time to access page tables can become less but the register is costly and very small as compared to the size of the page table. Therefore, this is also not a practical method.
- To overcome these drawbacks of paging, we need a memory that is cheaper than the register in cost and faster than the main memory to reduce the time taken by the CPU to access the page table again and again, and that will only focus on accessing the actual word.

- **Translation Lookaside buffer-** A Translation lookaside buffer (TLB) is defined as a memory cache hardware unit which is used to reduce the page table access time when page table is accessed again and again. TLB is a memory cache which is nearer to the CPU and the time taken to access TLB by CPU is less than that taken by CPU to access main memory. Or, we can say that TLB is smaller and faster than the main memory at the same time cheaper in cost and bigger in size than the CPU register.
- Performance of Paging-
  - Without TLB
    - Main memory access time = m
    - Effective memory access time (EMAT) = 2m
  - With TLB
    - TLB access time = c (where, c <<<<< m)
    - TLB hit ratio is a situation when the desired entry is found in TLB. If a hit happens then the CPU can simply access the actual address from the main memory. TLB hit ratio = Number of hits/ Total references
  - Effective memory access time (EMAT) = x(c+m) + (1-x) (c+2m) Where,
    - x → TLB hit,
    - c → time taken to access TLB,
    - m → time taken to access the main memory
    - k will be taken as 1, if single level paging has been used.
  - TLB miss- If the entry is not in TLB then it will be said to be a TLB miss, in this situation then the CPU has to access the page table from the main memory and then it will access the actual frame from the main memory.
- Virtual memory- Virtual Memory is a memory space where we can store large programs in the form of pages while their execution and only important pages or portions of processes are loaded into the main memory. It is a very useful technique as large virtual memory is provided for user programs even if the user's system has a very small physical memory.
  - The degree of Multiprogramming will be increased.
  - We can run large programs in the system, as virtual space is huge as compared to physical memory.
  - No need to purchase more memory RAMs.
  - More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

- Less I/O required, leads to faster and easy swapping of processes.
- The system becomes slower as page swapping takes time.
- Switching between different applications requires more time.
- Hard disk space will be shrinked and users cannot use it properly.
- Virtual memory is implemented by demand paging

- Performance of Virtual memory
  - Effective memory access time (EMAT) = P × s + (1- P) m
    - Where, Main memory access time= m
    - Page fault service time= s
    - Page fault rate = P
    - Page hit ratio = 1-P
- Demand Paging - This says keep all pages of a frame in the secondary memory space until they are required. Or, In other words, demand paging says that do not load any page in the main memory unless and until it is required. Whenever any page is referenced in the main memory for the first time, then we can find that page in the secondary memory.
- Segmentation- Segmentation is another non-contiguous memory allocation technique similar to paging. In segmentation, the process is not divided into fixed size pages like in paging.
  The process is divided into small modules for better visualization and implementation.
  - Segmentation do not suffer from internal fragmentation
  - Average Segment Size is greater than the actual page size.
  - It has less overhead.
  - Relocating segments is way easier than relocating entire address space.
  - The segment table is smaller in size as compared to the page table in paging.
  - It can suffer from external fragmentation.
  - It is tough to allocate contiguous memory to variable sized partitions.
  - Itt requires costly memory management algorithms.
- Difference between Paging and Segmentation

Paging	**Segmentation**
Non-Contiguous memory allocation	Non-contiguous memory allocation
Paging divides programs into fixed size pages.	Segmentation divides programs into variable size segments.
OS is responsible	Compiler is responsible.

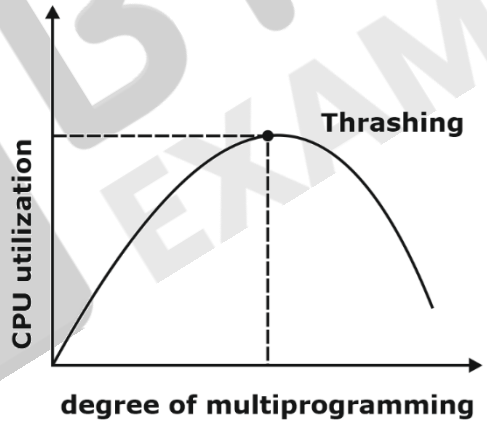
Paging	Segmentation
Paging is faster than segmentation	Segmentation is slower than paging
Paging is closer to Operating System	Segmentation is closer to User
It suffers from internal fragmentation	It suffers from external fragmentation
There is no external fragmentation	There is no external fragmentation
Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
Page table is used to maintain the page information.	Segment Table maintains the segment information
Page table entry has the frame number and some flag bits to represent details about pages.	Segment table entry has the base address of the segment and some protection bits for the segments.

- **Page replacement -** Page replacement is defined as a process of swapping out an existing page from the main memory frame and replacing it with the desired page. Page replacement is needed when, All the page frames of the main memory are already occupied. So, a page must be replaced to create a space for the desired page. Page replacement algorithms help to select which page should be swapped out of the main memory to create a space for the incoming required page.
- **Page replacement algorithm**
  - FIFO Page Replacement Algorithm
    - This algorithm works on the concept of "**First in First out** ".
    - This algorithm swaps out the oldest page from the main memory frame, oldest page means the page which is present in main memory for the longest time.
    - This algorithm is implemented by keeping track of all the pages in a queue.
    - Sometimes, on increasing the number of page frames in the main memory, the number of page faults also increases. This phenomenon is known as Belay's Anomaly
  - LIFO Page Replacement Algorithm
    - This algorithm works on the concept of "**Last in First out** ".
    - This algorithm swaps out the newest page from the main memory frame, newest page means the page which is present in main memory for the shortest time

- This algorithm is implemented by keeping track of all the pages in a stack.

- LRU Page Replacement Algorithm
  - This algorithm works on the concept of "**Least Recently Used** ".
  - This algorithm swaps out the page that has not been referred by the CPU for the longest time.
- Optimal Page Replacement Algorithm
  - Optimal Page Replacement algorithm swaps out the page that will not be referred by the CPU in future for the longest time span.
  - This algorithm cannot be implemented practically, because it is impossible to predict which pages will be used in the future
  - However, it is the best-known algorithm and a benchmark for other page replacement algorithms and this algorithm gives the least number of page misses.
- Random Page Replacement Algorithm
  - This algorithm randomly swaps out any page.
  - So, this algorithm can behave like any other page replacement algorithm like FIFO, LIFO, LRU, Optimal etc.

- **Thrashing**- If page fault and swapping of pages from memory happens very frequently at a higher rate, then the OS has to spend more time on swapping these pages. This state is known as thrashing. Due to this, CPU utilization will be reduced.



- Thrashing occurs when memory is over committed, and pages are tossed out while still needed. CPU utilization is very low when the page fault rate is high.

- **Inverted Paging**- Instead of using different or unique page tables, inverted paging says use a global page table. But it increases search time that's why this concept is not used. This technique can decrease load on main memory.

- **File System**
  - File system structure- A file system is used to allow the data to be stored, located, and retrieved easily. The file system itself is generally composed of many different levels, which are:
    - Application Programs
    - Logical File system

    - File organization module
    - Basic File System
    - I/O Control
    - Devices
  - Ways to access a file-
    - Sequential access- Bytes are accessed in order.
    - Random access- Bytes are accessed in any order.
  - A file system contains following major components -
    - Disk management organizes disk blocks into files.
    - Naming provides file names and directories to users, instead of track and sector numbers.
    - Protection keeps information secure from other users.
    - Reliability protects information loss due to system failure.
- Directory Structure
  - Single-level directory
  - Two-level directory structure
  - Tree structure directories
  - Acyclic graph directories
  - General graph directory
- **File**- A file is logically related records. From a programmer's perspective file is a data structure that has representation, definition, structure, operation, and attributes.
  - **Operations on File:** Create, Open, read, write, modify, seek, copy, move, close, rename, delete, etc.
  - **Attributes of Files**: name of file, size, owner, path, location, r/w information, last modification date, file creation date, etc.
- **Directory-** Directory is also a file which contains information about other files. It is a metadata of files.
- **File Usage Patterns**
  - Most files are small in size, and most file references are to small files.
  - Large files use up most of the disk space.
  - Large files account for most of the bytes transferred between memory and disk.
- Logical Structure of Disk
  - Partitions
    - Primary/Bootable- In which operating system and data can be stored. At Least one primary partition should be there on disk.
    - Secondary/ Non-Bootable One or more logical drives
- **File Control Block (FCB)**- It contains information about the file, including ownership, permission, location, etc.

- UNIX: UNIX file system (I node block or Indexed-node block)
- Windows: FAT, FAT 32, WTFS
- Linux: Extended File System

- File Allocation Methods:
  - Contiguous Allocation method-
    - Internal and external fragmentation exists.
    - Inflexible to increase size

      Random access
- Linked Allocation method-
  - Each file block on disk is associated with a pointer to the next block.
  - Flexibility to increase the file size.
  - Slow due to sequential access.
  - Suffers from external fragmentation.
  - MS-DOS file system, which uses the file Attribute Table (FAT) to implement linked-list files.
- Indexed Allocation method-
  - An index is used to directly track the file block locations.
  - No fragmentation
  - Flexibility to increase file size
  - Suffers from wasted space, supports direct access.
- Multilevel Indexed Allocation method-
  - Index entries point to index blocks as opposed to data blocks.
  - The file header, (i_ node data structure), holds index pointers.
  - First pointer points to data blocks.
  - A Single Indirect pointer points to a disk block which is a collection of pointers and further used as an index block. This is used when the file size is large, and we cannot index the entire file directly.
  - A Double Indirect Pointer is a pointer to disk block which is a set of pointers to another disk block and further pointing to index block. This is used when file size is extremely large and cannot be accessed directly with direct or single indirect pointers.
  - A Triple Indirect pointer is a pointer to disk block that is pointing to a collection of a disk block that is also pointing to a collection of a disk block and finally pointing to a separate index block which is further pointing to file block.
- Free Space Management
  - **Bit vector/ Bitmap**- Bit vector or bitmap often defined as the implementation of the free-space list. We can keep a bit vector as one bit per disk block. If a block is free, the corresponding bit is 1. Otherwise, bit is 0 (if the block is in use).
    - Memory requirements for storing bit vectors also increases with the increase in the disk size.

- Relatively simple and efficient to find the first free blocks or consecutive free blocks.
- To determine the first non-zero value, block number is required:

Block number = offset of first 1 bit + (Number of bits per word × Number of zero-value words)

- **Linked List** - With linked allocation, use existing links to form a free list. Link together all the free disk blocks. Space not wasted but It can be difficult to allocate contiguous blocks. List traversal is inefficient.
- **Grouping**- In the first free block we store the addresses of n free blocks. The first (n-1) of these blocks is free. The last block (nth) consists of addresses of another n blocks and so on. Addresses of so many free blocks can be found and accessed quickly.
- **Counting-** The Free-Space list can contain pairs (block number, count). Keep the address of the first free block number and the number n of free contiguous blocks that follow. Several contiguous blocks may be allocated or freed.

****