INTRODUCTION TO OPERATING SYSTEMS

An Operating System is a program that manages the Computer hardware. It controls and coordinates the use of the hardware among the various application programs for the various users.

A Process is a program in execution. As a process executes, it changes state

- New: The process is being created
- Running: Instructions are being executed
- Waiting: The process is waiting for some event to occur
- Ready: The process is waiting to be assigned to a process
- Terminated : The process has finished execution

Apart from the program code, it includes the current activity represented by

- Program Counter,
- Contents of Processor registers,
- Process Stack which contains temporary data like function parameters, return addresses and local variables
- Data section which contains global variables
- Heap for dynamic memory allocation

A Multi-programmed system can have many processes running simultaneously with the CPU multiplexed among them. By switching the CPU between the processes, the OS can make the computer more productive. There is Process Scheduler which selects the process among many processes that are ready, for program execution on the CPU. Switching the CPU to another process requires performing a state save of the current process and a state restore of new process, this is Context Switch.

Scheduling Algorithms

CPU Scheduler can select processes from ready queue based on various scheduling algorithms. Different scheduling algorithms have different properties, and the choice of a particular algorithm may favour one class of processes over another. The scheduling criteria include

- CPU utilization:
- Throughput: The number of processes that are completed per unit time.
- Waiting time: The sum of periods spent waiting in ready queue.
- Turnaround time: The interval between the time of submission of process to the time of completion.
- Response time: The time from submission of a request until the first response is produced.

The different scheduling algorithms are

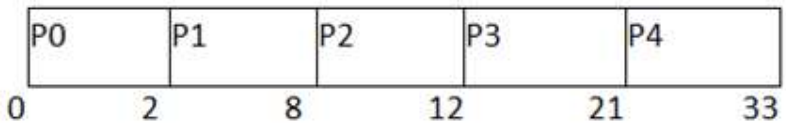
1. FCFS: First Come First Serve Scheduling

- It is the simplest algorithm to implement.
- The process with the minimal arrival time will get the CPU first.
- The lesser the arrival time, the sooner will the process gets the CPU.
- It is the non-pre-emptive type of scheduling.
- The Turnaround time and the waiting time are calculated by using the following formula.

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turnaround time - Burst Time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	8	4
3	3	9	21	18	9
4	4	12	33	29	17

Avg Waiting Time=31/5



2. SJF: Shortest Job First Scheduling

- The job with the shortest burst time will get the CPU first.
- The lesser the burst time, the sooner will the process get the CPU.
- It is the non-pre-emptive type of scheduling.
- However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.
- In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

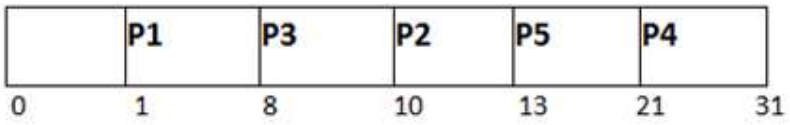
Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time

1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives)

.

- According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.
- Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.
- This will be executed till 8 units of time.
- Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.
- Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.



Avg Waiting Time = 27/5

3. SRTF: Shortest Remaining Time First Scheduling

- It is the pre-emptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution

4. Priority Scheduling

- In this algorithm, the priority will be assigned to each of the processes.
- The higher the priority, the sooner will the process get the CPU.
- If the priority of the two processes is same then they will be scheduled according to their arrival time.

5. Round Robin Scheduling

- In the Round Robin scheduling algorithm, the OS defines a time quantum (slice).
- All the processes will get executed in the cyclic way.
- Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a pre-emptive type of scheduling.

6. Multilevel Queue Scheduling

- A multi-level queue scheduling algorithm partitions the ready queue into several separate queues.
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm.

7. Multilevel Feedback Queue Scheduling

- Multilevel feedback queue scheduling, however, allows a process to move between queues.
- The idea is to separate processes with different CPU-burst characteristics.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.
- This form of aging prevents starvation.

Viva Questions

1. What is CPU Scheduler?
   Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
   CPU scheduling decisions may take place when a process:
   a. .Switches from running to waiting state.
   b. .Switches from running to ready state. c
   c. .Switches from waiting to ready.
   d. Terminates.
   Scheduling under a. and d. is non-pre-emptive.
   All other scheduling is pre-emptive

2. What are all the scheduling algorithms?
   a. FCFS(First Come First Serve)
   b. SJF(Shortest Job First)
   c. Round robin
   d. Priority Scheduling algorithms

3. Explain FCFS(First Come First Served)?
   a. The process that requests the CPU first is allocated the CPU first. The code for
   b. FCFS scheduling is simple to write and understand.
   c. Explain SJF(Shortest Job First)?
   d. The process which has the less burst time execute first. If both process have same burst time then FCFS will be used.

4. Explain Round Robin?
   The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. CPU switch between the processes based on a small unit of time called time slice.

5. Explain Priority Scheduling algorithm?
   CPU is allocated to the process with the highest priority.

6. Which algorithm gives minimum average waiting time?
   SJF(Shortest Job First)

7. What is CPU utilization?
   We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent.

8. What is Throughput?

The amount of work is being done by the CPU. One unit of work is the number of processes that are completed per unit time, called throughput

9. What is Turnaround time.
   The interval from the time of submission of a process to the time of completion is the turnaround time

10. What is waiting time?
    Waiting time is the sum of the periods spent waiting in the ready queue.

11. What is Response time?
    the time from the submission of a request until the first response is produced.

12. What are short, long and medium-term scheduling?
    a. Long term scheduler determines which programs are admitted to the system for processing. It controls the degree of multiprogramming. Once admitted, a job becomes a process.
    b. Medium term scheduling is part of the swapping function. This relates to processes that are in a blocked or suspended state. They are swapped out of real-memory until they are ready to execute. The swapping-in decision is based on memory-management criteria.
    c. Short term scheduler, also known as a dispatcher executes most frequently, and makes the finest-grained decision of which process should execute next. This scheduler is invoked whenever an event occurs. It may lead to interruption of one process by pre-emption.

13. What are turnaround time and response time?
    Turnaround time is the interval between the submission of a job and its completion.

14. What is pre-emptive and non-pre-emptive scheduling?
    a. Pre-emptive scheduling: The pre-emptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.
    b. Non-Pre-emptive scheduling: When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.

DEADLOCK

Deadlock :

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause (including itself).

Waiting for an event could be:

- Waiting for access to a critical section
- Waiting for a resource Note that it is usually a non-pre-emptable (resource). Pre-emptable resources can be yanked away and given to another.

Conditions for Deadlock

- Mutual exclusion: resources cannot be shared.
- Hold and wait: processes request resources incrementally, and hold on to what they've got.
- No pre-emption: resources cannot be forcibly taken from processes.
- Circular wait: circular chain of waiting, in which each process is waiting for a resource held by the next process in the chain.

Deadlock Avoidance

- This approach to the deadlock problem anticipates deadlock before it actually occurs.
- This approach employs an algorithm to access the possibility that deadlock could occur and acting accordingly.
- This method differs from deadlock prevention, which guarantees that deadlock cannot occur by denying one of the necessary conditions of deadlock.
- If the necessary conditions for a deadlock are in place, it is still possible to avoid deadlock by being careful when resources are allocated.
- Perhaps the most famous deadlock avoidance algorithm, due to Dijkstra [1965], is the Banker's algorithm.

Safe State

Safe state is one where

- It is not a deadlocked state
- There is some sequence by which all requests can be satisfied.

- To avoid deadlocks, we try to make only those transitions that will take you from one safe state to another.

- We avoid transitions to unsafe state (a state that is not deadlocked, and is not safe).
- Banker's algorithm is a **deadlock avoidance algorithm**.
- It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.
- Consider there are n account holders in a bank and the sum of the money in all of their accounts is S.
- Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has.
- Then it checks if that difference is greater than S.
- It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.
- Banker's algorithm works in a similar way in computers.
- The Banker's algorithm is run by the operating system whenever a process requests resources.
- The algorithm prevents deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur).
- When a new process enters a system, it must declare the maximum number of instances of each resource type that may not exceed the total number of resources in the system.
- For the Banker's algorithm to work, it needs to know three things:
- How much of each resource each process could possibly request
- How much of each resource each process is currently holding
- How much of each resource the system has available
- Some of the resources that are tracked in real systems are memory, semaphores and interface access.

Viva questions

1. What is deadlock?
   Deadlock is a situation that when two or more process waiting for each other and holding the resource which is required by another process.

2. What are the necessary conditions to occur deadlock?
   Mutual exclusion: At least one resource must be held in a non-sharable mode, that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
   Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
   No pre-emption: Resources cannot be pre-empted.; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
   Circular wait: A set {P$, Pi, ..., Pn} of waiting processes must exist such that P-0 is waiting for a resource held by P\, P\ is waiting for a resource held by P?, •••, P.,--i is waiting for a resource held by Pn, and P, is waiting for a resource held by Pn.

3. Explain about resource allocation graph?
   Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

4. What are the methods to handle the dead locks?
   a. We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
   b. We can allow the system to enter a deadlock state, detect it, and recover.
   c. We can ignore the problem altogether and pretend that deadlocks never occur in the system.
   d. The third solution is the one used by most operating systems

5. What are the deadlock avoidance algorithms?
   A dead lock avoidance algorithm dynamically examines there source-allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources, and the maximum demand of the process. There are two algorithms:
   Resource allocation graph algorithm
   a. Banker's algorithm
   b. Safety algorithm
   c. Resource request algorithm

6. What is Bankers Algorithm.
   It is an algorithm which used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

7. What is a Safe State and what is its use in deadlock avoidance?
   When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. System is in safe state if there exists a safe sequence of all processes. Deadlock Avoidance: ensure that a system will never enter an unsafe state.

8. What is starvation and aging?
   Starvation is Resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.

9. What is a Safe State and its' use in deadlock avoidance?
   When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
   - System is in safe state if there exists a safe sequence of all processes.
   - Sequence is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished. When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate. When Pi terminates, Pi+1 can obtain its needed resources, and so on.
   - Deadlock Avoidance Þ ensure that a system will never enter an unsafe state.

10. Recovery from Deadlock?
    - Process Termination:
      ->Abort all deadlocked processes.
      ->Abort one process at a time until the deadlock cycle iseliminated.
      ->In which order should we choose to abort?
    - Priority of the process.
      How long process has computed, and how much longer tocompletion.
      Resources the process has used.
      Resources process needs to complete.
      How many processes will need to be terminated?
      Is process interactive or batch?
    - Resource Preemption:
      ->Selecting a victim – minimize cost.
      ->Rollback – return to some safe state, restart process for thatstate.
      ->Starvation – same process may always be picked as victim,include number of rollback in cost factor.

DISK SCHEDULING

Disk scheduling is is done by operating systems to schedule I/O requests arriving for disk. It is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

- Seek Time:Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- Rotational Latency: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- Disk Access Time: Disk Access Time is:

  Disk Access Time = Seek Time + Rotational Latency + Transfer Time

- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

- **Disk Scheduling Algorithms**
  - FCFS
  - SSTF
  - SCAN
  - CSCAN
  - LOOK
  - CLOOK

1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

   Advantages:
   - Every request gets a fair chance
   - No indefinite postponement

   Disadvantages:
   - Does not try to optimize seek time
   - May not provide the best possible service

3. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

   Advantages:
- High throughput
- Low variance of response time
- Average response time

   Disadvantages:
   - Long waiting time for requests for locations just visited by disk arm. These situations are avoided in *CSAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

   Advantages:
   - Provides more uniform wait time compared to SCAN

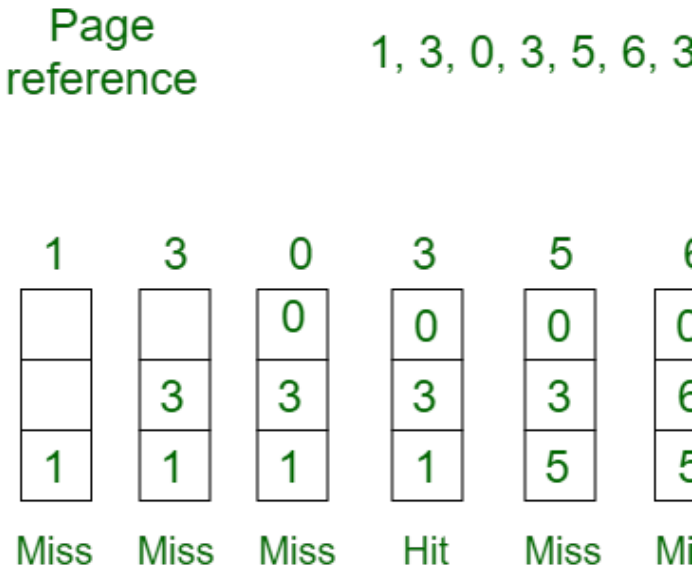
4. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

PAGE REPLACEMENT TECHNIQUES

First In First Out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

- **Example-1**Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of page faults.



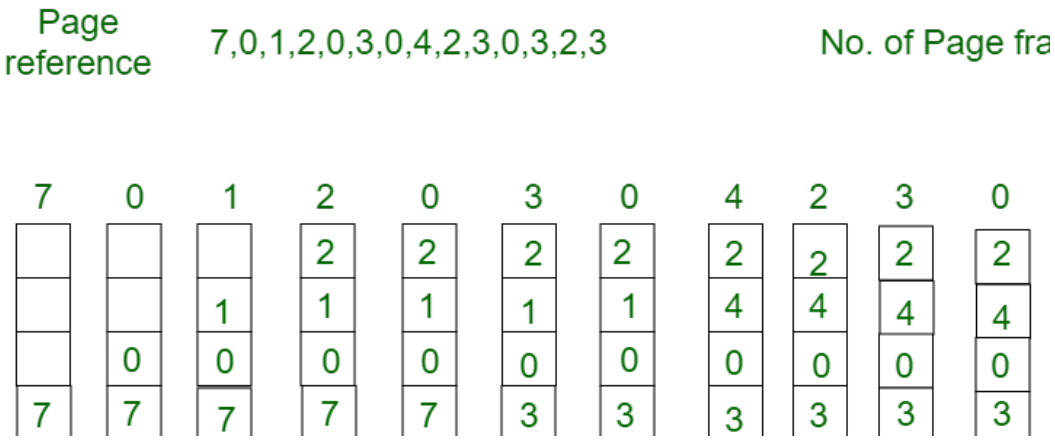
- Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
  when 3 comes, it is already in memory so —> **0 Page Faults.**
  Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
  6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
  Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

Optimal Page replacement –

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

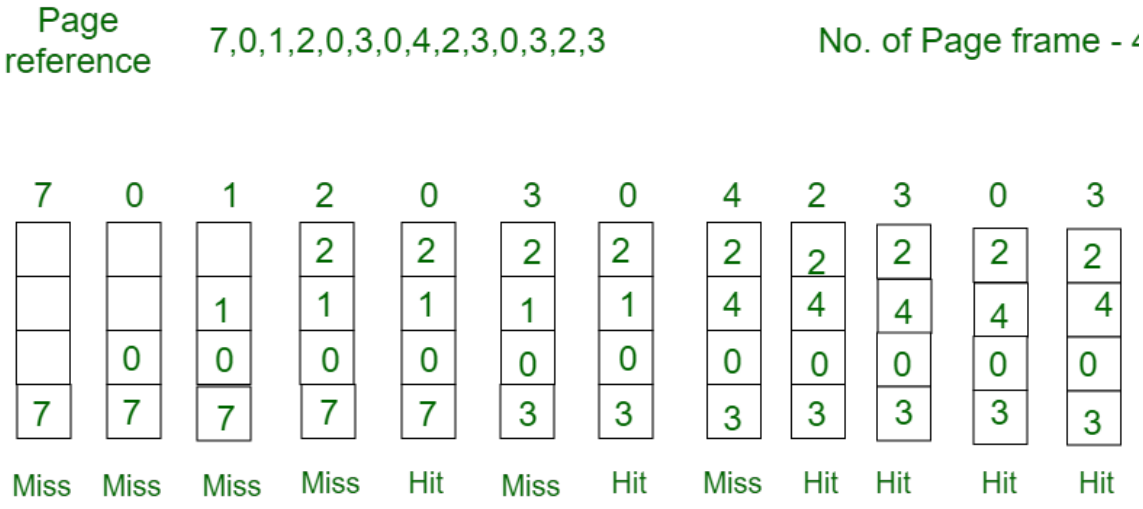


- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
  0 is already there so —> **0 Page fault.**
  when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
  0 is already there so —> **0 Page fault.**.
  4 will takes place of 1 —> **1 Page Fault.**
- Now for the further page reference string —> **0 Page fault** because they are already available in the memory.
- Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

Least Recently Used –

In this algorithm page will be replaced which is least recently used.

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.



- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
  0 is already their so —> **0 Page fault.**
  when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
  0 is already in memory so —> **0 Page fault**.
  4 will takes place of 1 —> **1 Page Fault**
  Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Viva Questions

1. Why paging is used?
   Paging is solution to external fragmentation problem which is to permit the logical address space of a process to be non-contiguous, thus allowing a process to be allocating physical memory wherever the latter is available.

2. What is virtual memory?
   Virtual memory is memory management technique which is used to execute the process which has more than actual memory size.

3. What is Demand Paging?
   It is memory management technique used in virtual memory such that page will not load into the memory until it is needed.

4. What are all page replacement algorithms?
   a. FIFO(First in First out)
      2. Optimal Page Replacement
      3. LRU(Least-Recently-used)

5. Which page replacement algorithm will have less page fault rate?
   Optimal Page Replacement
6. What is thrashing?
   It is situation that CPU spends more time on paging than executing.

7. What is swapping
   A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. This process is called swapping.

8. What is fragmentation?
   fragmentation is a phenomenon in which storage space is used inefficiently, reducing capacity or performance.

9. Explain External fragmentation?
   As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous.

10. Explain Internal fragmentation?
    Consider a multiple-partition allocation scheme with a hole of 18,464 bytes. Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes. The overhead to keep track of this hole will be substantially

larger than the hole itself. The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size. With this approach, the memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation.

11. What is paging?
    Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous. Paging avoids the considerable problem of fitting memory chunks of varying sizes onto the backing store.

12. What is frame?
    Breaking main memory into fixed number of blocks called frames.

13. What is page?
    Breaking logical memory into blocks of same size is page.

14. What is the best page size when designing an operating system?
    The best paging size varies from system to system, so there is no single best when it comes to page size. There are different factors to consider in order to come up with a suitable page size, such as page table, paging time, and its effect on the overall efficiency of the operating system.

15. What is virtual memory?
    Virtual memory is hardware technique where the system appears to have more memory that it actually does. This is done by time-sharing, the physical memory and storage parts of the memory one disk when they are not actively being used.

16. What is Throughput, Turnaround time, waiting time and Response time?
    Throughput – number of processes that complete their execution per time unit. Turnaround time – amount of time to execute a particular process. Waiting time – amount of time a process has been waiting in the ready queue. Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment).

17. Explain Belady's Anomaly?
    Also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns.

18. What is fragmentation? Different types of fragmentation?

Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy any request.

- External Fragmentation: External Fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effectively used. If too much external fragmentation occurs, the amount of usable memory is drastically reduced. Total memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation: Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks. Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used Reduce external fragmentation by compaction
  ->Shuffle memory contents to place all free memory together in one large block.
  ->Compaction is possible only if relocation is dynamic, and is done at execution time.

19. Explain Segmentation with paging?
    Segments can be of different lengths, so it is harder to find a place for a segment in memory than a page. With segmented virtual memory, we get the benefits of virtual memory but we still have to do dynamic storage allocation of physical memory. In order to avoid this, it is possible to combine segmentation and paging into a two-level virtual memory system. Each segment descriptor points to page table for that segment. This give some of the advantages of paging (easy placement) with some of the advantages of segments (logical division of the program).

20. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs?
    A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted

FILE ORGANISATION TECHNIQUES

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory :

Name : The name visible to user.
Type : Type of the directory.
Location : Device and location on the device where the file header is located.
Size : Number of bytes/words/blocks in the file.
Position : Current next-read/next-write pointers.
Protection : Access control on read/write/execute/delete.
Usage : Time of creation, access, modification etc.
Mounting : When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.

Advantages of maintaining directories are:

Efficiency: A file can be located more quickly.
Naming: It becomes convenient for users as two users can have same name for different files or may have different name for same file.
Grouping: Logical grouping of files can be done by properties e.g. all java programs, all games etc.
Naming problem: Users cannot have same name for two files.
Grouping problem: Users cannot group files according to their need.

Two-Level Directory

In this separate directories for each user is maintained.
Path name: Due to two levels there is a path name for every file to locate that file.
So same file name for different user are possible. Searching is efficient in this method.

Single-Level Directory

In this a single directory is maintained for all the users.

ASSEMBLER

- System software which is used to convert assembly language program to its equivalent object code. Input to the assembler is a source code written in assembly language. Output is the object code. Design of assembler depends upon the machine architecture as the language used is mnemonic language.

Analysis Phase

- Build the Symbol table.
- Separate labels, opcodes and operand fields in a statement.
- Check correctness of opcodes by looking at the contents of the mnemonics table.
- Update contents of location counter based on the length of each instruction.

Synthesis Phase

- Look at the mnemonics table and get the opcode corresponding to the mnemonic.
- Obtain the address of a memory operand from the symbol table.
- Synthesize the machine instruction.

TYPES OF ASSEMBLER

1. Single Pass Assembler
2. Two Pass Assembler

Single Pass Assembler

- The assembler reads the source file once.
- During the single pass, the assembler handles both label definitions and assembly.
- Here whole process of scanning, parsing and object code conversion is done in single pass.

- The only problem with this method is resolving forward reference.
- One pass assembler is used when it is necessary or desirable to avoid a second pass over the source program.
- The external storage for the intermediate file between two passes is slow or is inconvenient to use.
- One-pass/ Single pass assemblers are used when
    - It is necessary or desirable to avoid a second pass over the source program.
    - The external storage for the intermediate file between two passes is slow or is inconvenient to use
- Main problem: forward references to both data and instructions
    - One simple way to eliminate this problem: require that all areas be defined before they are referenced.
    - It is possible, although inconvenient, to do so for data items.
    - Forward jump to instruction items cannot be easily eliminated.

Two Pass Assembler

- Here there are two passes
- It resolves the forward references and then converts in to the object code.
- Here forward references in symbol definition are not allowed.
- Symbol definition must be completed in pass 1.

  (Forward reference: When we use the symbol or literal (identifier) before declaring it and the error caused due to this is called a **Forward Reference** Problem. For example:- int c, b=10;)
- In the first pass it reads the entire source file, looking only the label definitions.
- All labels are collected, assigned values and placed in the symbol table in this pass.
- No instructions are assembled and at the end of the pass, the symbol table should contain all the labels defined in the program.
- In the second pass, the instructions are again read and are assembled using the symbol table.

Pass 1 (Define Symbols):

i. Assign address to all statements in program

ii. Save the values assigned to all labels for use in pass 2.

iii. Perform some processing of assembler functions

Pass 2 (Assemble Instructions and Generate Object Code):

i. Assembler instructions.

ii. Generate data values defined by BYTE, WORD, etc.

iii. Perform processing of assembler directives not done during pass 1.

iv. Write object program and assembly listing.

1. Define the basic functions of assembler.
* Translating mnemonic operation codes to their machine language equivalents.
* Assigning machine addresses to symbolic labels used by the programmer.

2. What is meant by assembler directives? Give example.
These are the statements that are not translated into machine instructions, but they provide instructions to assembler itself.
example START,END,BYTE,WORD,RESW and RESB.
3. What are forward references?
It is a reference to a label that is defined later in a program.
Consider the statement
10 1000 STL RETADR
. . . .
. . . .
80 1036 RETADR RESW 1
The first instruction contains a forward reference RETADR. If we attempt to translate the program line by line, we will unable to process the statement in line10 because we do not know the address that will be assigned to RETADR .The address is assigned later(in line 80) in the program.

4. What are the three different records used in object program?
The header record, text record and the end record are the three different records used in object program.
The header record contains the program name, starting address and
length of the program.
Text record contains the translated instructions and data of the program.
End record marks the end of the object program and specifies the address in the program where execution is to begin.

5. What is the need of SYMTAB (symbol table) in assembler?
The symbol table includes the name and value for each symbol in the
source program, together with flags to indicate error conditions. Some times it may contain details about the data area. SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval.

6. What is the need of OPTAB (operation code table) in assembler?
The operation code table contains the mnemonic operation code and its
machine language equivalent. Some assemblers it may also contain information about instruction format and length. OPTAB is usually organized as a hash table, with mnemonic operation code as the key.

10. Write the steps required to translate the source program to object program.
• Convert mnemonic operation codes to their machine language

equivalents.
• Convert symbolic operands to their equivalent machine addresses
• Build the machine instruction in the proper format.
• Convert the data constants specified in the source program into their internal machine representation
• Write the object program and assembly listing.

11. What is the use of the variable LOCCTR (location counter) in assembler?
This variable is used to assign addresses to the symbols. LOCCTR is initialized to the beginning address specified in the START statement. After each source statement is processed the length of the assembled instruction or data area to be generated is added to LOCCTR and hence whenever we reach a label in the source program the current value of LOCCTR gives the address associated with the label.
12. Define load and go assembler.
One pass assembler that generates their object code in memory for immediate execution is known as load and go assembler. Here no object programmer is written out and hence no need for loader.

13. What are the two different types of jump statements used in MASM assembler?
• Near jump
A near jump is a jump to a target in the same segment and it is assembled by using a current code segment CS.
• Far jump
A far jump is a jump to a target in a different code segment and it is assembled by using different segment registers .

15. Differentiate the assembler directives RESW and RESB.
RESW –It reserves the indicated number of words for data area.
Eg: 10 1003 THREE RESW 1
In this instruction one word area (3 bytes) is reserved for the symbol THREE. If the memory is byte addressable then the address assigned for the next symbol is 1006.
RESB –It reserves the indicated number of bytes for data area.
Eg: 10 1008 INPUT RESB 1
In this instruction one byte area is reserved for the symbol INPUT .Hence the address assigned for the next symbol is 1009.

17. Write down the pass numbers (PASS 1/ PASS 2) of the following activities that occur in a two pass assembler:
a. Object code generation
b. Literals added to literal table
c. Listing printed
d. Address location of local symbols

Answer:

a. Object code generation - PASS 2
b. Literals added to literal table – PASS 1
c. Listing printed – PASS2
d. Address location of local symbols – PASS1

18. What is meant by machine independent assembler features?

The assembler features that do not depend upon the machine architecture are known as machine independent assembler features.
Eg: program blocks, Literals.

20. What is meant by external references?

Assembler program can be divided into many sections known as control sections and each control section can be loaded and relocated independently of the others. If the instruction in one control section need to refer instruction or data in another control section.the assembler is unable to process these references in normal way. Such references between control are called external references.

25. What is the use of the assembler directive START?

The assembler directive START gives the name and starting address of the program.
The format is
PN START 1000
Here
PN – Name of the program
1000 - Starting address of the program.

26. What are the basic functions of loaders?

- Loading – brings the object program into memory for execution
- Relocation – modifies the object program so that it can be loaded at an address different from the location originally specified
- Linking – combines two or more separate object programs and also supplies the information needed to reference them.

LOADER AND LINKER

- The source program written in assembly language or high level language will be converted to object program, which is in the machine language form for execution.
- This conversion is either from assembler or from compiler, contains translated instructions and data values from the source program, or specific addresses in primary memory where these items are to be loaded for execution.
- This contain three processes:
    1. Loading- It allocates memory location and brings the object program in to memory for execution.
    2. Linking- It combines two or more separate object programs and supplies the information needed to allow references between them.
    3. Relocation- It modifies the object program so that it can be loaded at address different from the location originally specified.

LOADER: It is a utility of an operating system. It copies program from a storage device to a computer's main memory, where the program can then be executed.

Various Steps Loader Performs

1. Read executable file's header to determine the size of text and data segments.
2. Create new address space for the program.
3. Copies instructions and add data in to address space.
4. Copies arguments passed to the program on the stack.
5. Initializes the machine registers including the stack pointer.
6. Jumps to a start-up routine that copies the program's arguments from the stack to registers and calls the program's main routine.

Types of Loader

1. Assemble and Go Loader
2. Relocating Loader (Relative Loader)
3. Absolute Loader (Bootstrap Loader)
4. Direct Linking Loader

ABSOLUTE LOADER

- It is also known as Bootstrap Loader.

- It is the simplest loader.
- It can read a machine language program from the specified back up storage and place it in memory starting from a pre- determined address.
- Machine language program so loaded will work correctly only if it is loaded starting from the specified address.
- Absolute type of loader is impractical, there are lots of complications involved in loading the program.
- "Bootstrap loader" is an example of absolute loader.

Advantage:

- It simply performs input and output operation to load a program into the main memory.
- It is coded in very few machine instructions.
- Program is stored in the library in their ready to execute form. Such a library is called a Phase Library.

Disadvantage:

- Programmer must explicitly specify the assembler the memory where the program is to be loaded.
- Handling multiple subroutine become difficult since the programmer must specify the address of the routines whenever they are referenced to perform subroutine linkage.
- When dealing with lots of subroutines the manual shuffling and re-shuffling of memory address references in the routines become tedious and complex.

Design of Absolute Loader

- The operation of absolute loader is simple.
- Object code is loaded to specified locations in the memory.
- At the end the loader jumps to the specified address to begin execution of the loaded program.
- Initially the header record is checked to verify that the correct program has been presented for loading
- As each text record is read the object code it contains is moved to the indicated memory location.

When the end record is encountered loader jumps to the specified i.e. location starting location of the program to begin execution.

SIMPLE BOOTSTRAP LOADER

- It is a special type of absolute loader that is executed when computer is first turned o or restarted.
- The bootstrap loads the first program to be run by the computer- usually by operating systems.

Bootstrap Loader for SIC/XE

- The bootstrap begins at address 0 in the memory of the machine.
- It loads the operating system starting at address 80.
- Because this loader is used in a unique situation, the program to be loaded can be represented in very simple format:
  i. Each byte of object code to be loaded is represented on device F1 as two hexadecimal digits.
  ii. The object code from device F1 is always loaded into consecutive bytes of memory, starting at address 80.
  iii. After loading, the bootstrap jumps to address 80 to execute loaded program

Algorithm

- Clear the accumulator content.
- The index register 'X' is initialized to the hexadecimal value of 80.
- Test the input device to see if it is ready.
- When the input device becomes ready, read an ASCII character code.
- The input characters that have ASCII code less than hexadecimal 30 is skipped which will prevent the bootstrap, from misinterpreting any control bytes as end of file marker.
- Convert the ASCII character code to hexadecimal digit.
- Save the hexadecimal digit in register 'S' and left shift it 4 bit position.
- Repeat the processing from step 4 to 6 to get the next character from the input device and convert it to hexadecimal form.
- The hexadecimal value of the 2nd character read is added with the left shifted hexadecimal value of the 1st character which is already stored in register 'S'.
- The resultant byte is stored in the address currently in register 'X'.
- Increment the value of index register by 1, to make it hold the next address location
- Repeat steps 3 to 11 until an end of the file is encountered.
- If the character read indicate the end of the file, jump to the starting location of the program just loaded to begin the program execution.

Repeat the steps from 3 to 13 until there is no input

MACRO PROCESSORS

A macro instruction (macro) is a notational convenience for the programmer. It allow the programmer to write a shorthand version of a program . A macro represents a commonly used group of statements in the source programming language. It replaces each macro instruction with the corresponding group of source language statements.

A macro processor Essentially involve the substitution of one group of characters or lines for another. Normally, it performs no analysis of the text it handles. It doesn't concern the meaning of the involved statements during macro expansion The design of a macro processor generally is machine independent.

Macro processor should processes the

- Macro definitions : Define macro name, group of instructions
- Macro invocation (macro calls): A body is simply copied or substituted at the point of call

Two new assembler directives are used in macro definition:
MACRO: identify the beginning of a macro definition
MEND: identify the end of a macro definition

```
label       op      operands
name    MACRO   parameters
:
body
:
MEND
```

Parameters: the entries in the operand field identify the parameters of the macro instruction . We require each parameter begins with '&'
Body: the statements that will be generated as the expansion of the macro.
Prototype for the macro: The macro name and parameters define a pattern or prototype for the macro instructions used by the programmer

One-pass macro processor
Two-pass macro processor

- All macro definitions are processed during the first pass.
- All macro invocation statements are expanded during the second pass.

Nested macro definitions - The body of a macro contains definitions of other macros because all macros would have to be defined during the first pass before any macro invocations were expanded.

VIVA QUESTIONS

1. Define macro processor.
Macro processor is system software that replaces each macro instruction with the corresponding group of source language statements. This is also called as expanding of macros.

2. What do macro expansion statements mean?
These statements give the name of the macro instruction being invoked
and the arguments to be used in expanding the macros. These statements are also known as macro call.

3. What are the directives used in macro definition?
MACRO - it identifies the beginning of the macro definition
MEND - it marks the end of the macro definition

4. What are the data structures used in macro processor?
DEFTAB – the macro definitions are stored in a definition table i.e. it contains a macro prototype and the statements that make up the macro body.
NAMTAB – it is used to store the macro names and it contains two
pointers for each macro instruction which indicate the starting and end location of macro definition in DEFTAB. it also serves as an index to DEFTAB
ARGTAB – it is used to store the arguments during the expansion of macro invocations.

5. Define conditional macro expansion.
If the macro is expanded depends upon some conditions in macro
definition (depending on the arguments supplied in the macro expansion) then it is called as conditional macro expansion.

6. What is the use of macro time variable?
Macro time variable can be used to store working values during the macro expansion. Any symbol that begins with the character & and then is not a macro instruction parameter is assumed to be a macro time variable.

7. What are the statements used for conditional macro expansion?
IF-ELSE-ENDIF statement
WHILE-ENDW statement

8. What is meant by positional parameters?
If the parameters and arguments were associated with each other
according to their positions in the macro prototype and the macro invocation statement, then these parameters in macro definitions are called as positional parameters.

10. What are known as nested macro call?
The statement, in which a macro calls on another macro, is called nested macro call. In the nested macro call, the call is done by outer macro and the macro called is the inner macro.

11. How the macro is processed using two passes?
Pass1: processing of definitions
Pass 2:actual-macro expansion.

12. Give the advantage of line by line processors.
• It avoids the extra pass over the source program during assembling.
• It may use some of the utility that can be used by language translators so that can be loaded once.

13. What is meant by line by line processor?
This macro processor reads the source program statements, process the
statements and then the output lines are passed to the language translators as they are generated, instead of being written in an expanded file.

14. Give the advantages of general-purpose macro processors.
• The programmer does not need to learn about a macro facility for each compiler.
• Overall saving in software development cost and maintenance cost.

15. What is meant by general-purpose macro processors?
The macro processors that are not dependent on any particular
programming language, but can be used with a variety of different languages are known as general purpose macro processors.
Eg. The ELENA macro processor.

16. What are the important factors considered while designing general purpose macro processors?
• comments
• grouping of statements
• tokens
• syntax used for macro definitions

18. How the nested macro calls are executed?
The execution of nested macro call follows the LIFO rule. In case of nested macro calls the expansion of the latest macro call is completed first.

19. Mention the tasks involved in macro expansion.
• identify the macro calls in the program
• the values of formal parameters are identified
• maintain the values of expansion time variables declared in a macro
• expansion time control flow is organized
• determining the values of sequencing symbols

• expansion of a model statement is performed

20. How to design the pass structure of a macro assembler?
To design the structure of macro-assembler, the functions of macro pre-processor and the conventional assembler are merged. After merging, the functions are structured into passes of the macro assembler.