Question Bank Soln For ETE OS

BY JAAT

1. a) FIFO replacement

b) LRU replacement

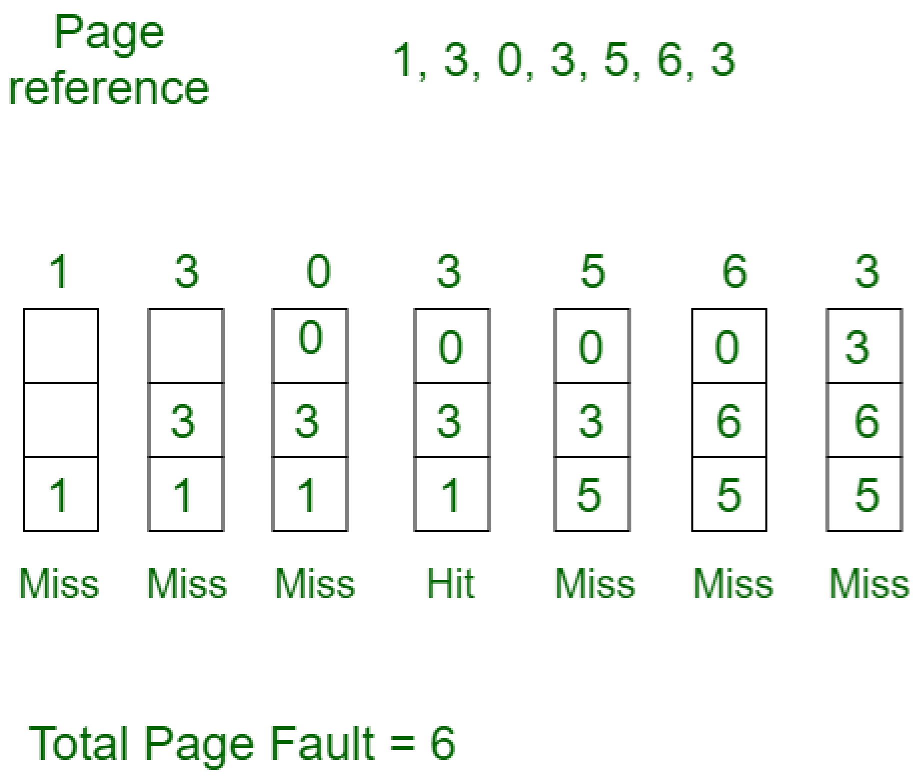
c)Optimal Page Replacement.-10

ANS In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms:
1. 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, 3 with 3 page frames.Find the number of page faults.

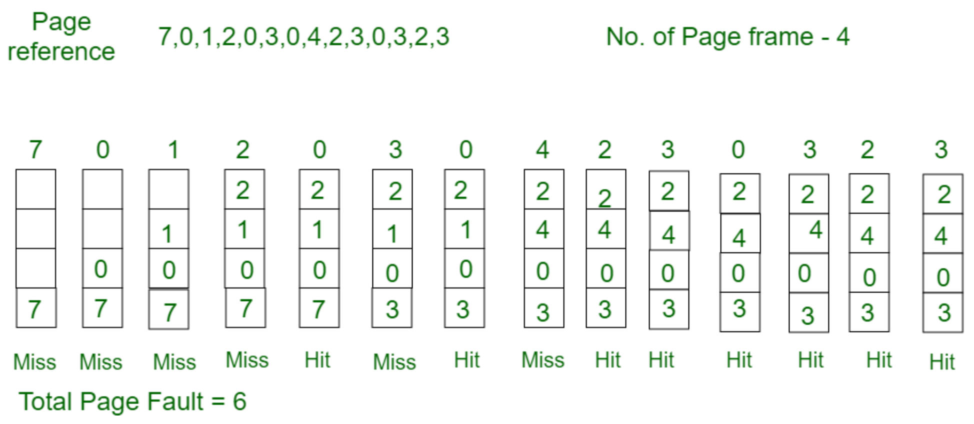


Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
when 3 comes, it is already in memory so —> **0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.** 6 comes, it is also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

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 strings 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.

2. 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, 3 with 4 page frame. Find number of page fault.

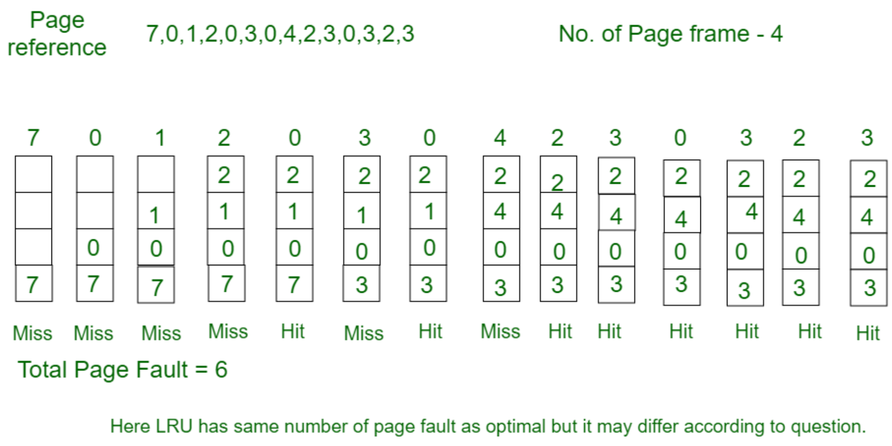


Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.** when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.** 0 is already there so —> **0 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.

3. 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, 3 with 4 page frames. Find number of page faults.



Page reference
7,0,1,2,0,3,0,4,2,3,0,3,2,3
No. of Page frame - 4
Miss Miss Miss Miss Hit Miss Hit Miss Hit Hit Hit Hit Hit Hit
Total Page Fault = 6
Here LRU has same number of page fault as optimal but it may differ according to question.

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.

2. FCFS, SJF, RR-15

ANS 1. Shortest Job First (SJF) **:**
Shortest Job First (SJF) Scheduling Algorithm is based upon the burst time of the process. The processes are put into the ready queue based on their burst times. In this algorithm, the process with the least burst time is processed first. The burst time of only those processes is compared that are present or have arrived until that time. It is also non-preemptive in nature. Its preemptive version is called Shortest Remaining Time First (SRTF) algorithm.

The major advantage of this algorithm is that it gives the minimum waiting time for a given set of processes and thus reduces the average waiting time. The disadvantage of this algorithm is that long processes may never be processed by the system and may remain in the queue for very long time leading to starvation of processes.

Note –
If two processes have same burst time then the tie is broken using FCFS, i.e., the process that arrived first is processed first.

2.Round-Robin (RR) **:**
Round-Robin (RR) Scheduling Algorithm is particularly designed for time sharing systems. The processes are put into the ready queue which is a circular queue in this case. In this case a small unit of time known as time quantum is defined. The algorithm selects the first process from the queue and executes it for the time defined by the time quantum. If the process has burst time less than the time quantum then the CPU executes the next process but if it has burst time higher than the time quantum then the process is interrupted and next process is executed for same time quantum. If a process is interrupted then a context switch happens and the process is put back at the tail of the queue. It is preemptive in nature.

This algorithm mainly depends on the time quantum. Very large time quantum makes RR same as the FCFS while a very small time quantum will lead to the overhead as context switch will happen again and again after very small intervals.

The major advantage of this algorithm is that all processes get executed one after the other which does not lead to starvation of processes or waiting by process for quite long time to get executed.

3. FCFS stands for **First Come First Serve**. In the FCFS scheduling algorithm, the job that arrived first in the ready queue is allocated to the CPU and then the job that came second, and so on. We can say that the ready queue acts as a FIFO (First In First Out) queue thus the arriving jobs/processes are placed at the end of the queue.

FCFS is a non-preemptive scheduling algorithm as a process holds the CPU until it either terminates or performs I/O. Thus, if a longer job has been assigned to the CPU then many shorter jobs after it will have to wait. This algorithm is used in most batch operating systems.

Characteristics:

- It follows the non-preemptive approach i.e. once a process has control over the CPU it will not preempt until it terminates.
- The criteria for the selection of processes is arrival time. The dispatcher selects the first job in the ready queue and this job runs to completion of its CPU burst.
- The average waiting time is very high so not optimal and thus gives poor performance.
- Simple and easy to understand: FCFS is a simple and easy-to-understand scheduling algorithm that does not require complex computations or heuristics.
- Non-preemptive approach: As mentioned, FCFS follows the non-preemptive approach, which means that once a process has control over the CPU, it will continue to run until it completes its CPU burst or voluntarily relinquishes the CPU.
- Queue management: FCFS uses a simple queue structure to manage the order in which processes are scheduled. The first process that arrives is added to the back of the queue, and subsequent processes are added to the end of the queue.

- Starvation: FCFS can lead to the problem of starvation, where a long-running process can cause other processes to wait for an extended period, leading to poor performance.
- Average turnaround time: FCFS can have a high average turnaround time, as longer processes may have to wait for shorter processes to complete, leading to longer waiting times.
- Limited scheduling flexibility: FCFS has limited scheduling flexibility, as it does not take into account the priority or resource requirements of the processes. This can lead to suboptimal performance in certain scenarios, such as when there are both CPU-bound and I/O-bound processes.
- Not optimal for interactive systems: FCFS is not optimal for interactive systems, as it may cause long waiting times for users, leading to a poor user experience.

Advantages:

- FCFS algorithm is simple, easy to implement into any preexisting system, and easy to understand.
- Better for processes with large burst time as there is no context switch involved between processes.
- It is a fair algorithm as priority is not involved, processes that arrive first get served first.

Disadvantages:

- Convoy effect occurs i.e. all small processes have to wait for one large process to get off the CPU.
- It is non-preemptive, the process will not release the CPU until it finishes its task and terminates.
- It is not appropriate for interactive systems as it cannot guarantee a short response time.
- Average waiting time is high and the turnaround time is unpredictable which leads to poor performance

3. FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK-15
ANS FCFS (First-Come, First-Served):

FCFS is a disk scheduling algorithm in which the requests are processed in the order they arrive. It follows a simple principle of serving the requests based on their arrival time. The first request that arrives is the first to be served.

Advantages:

1. Simple and easy to implement.
2. No starvation, as every request gets a chance to be served.

Disadvantages:

1. Poor performance in terms of seek time.
2. Longer waiting times for requests that arrive later.
3. Inefficient use of disk bandwidth.

SSTF (Shortest Seek Time First):

SSTF is a disk scheduling algorithm that selects the request with the shortest seek time from the current head position. The seek time is the time required to move the disk arm to the requested track.

Advantages:

1. Reduces the overall seek time compared to FCFS.
2. Provides better response time for requests.

Disadvantages:

1. May cause starvation for requests located far from the current head position.

2. May result in increased disk arm movement due to selecting requests only based on seek time.

SCAN:

SCAN is a disk scheduling algorithm that moves the disk arm from one end of the disk to the other, serving requests along the way. Once it reaches the end, it reverses direction and serves the remaining requests on the other side.

Advantages:

1. Provides a more balanced disk access by serving requests from both ends.

2. No starvation for any request.

Disadvantages:

1. May cause higher waiting time for requests located in the middle of the disk.

2. The head movement can be inefficient if most requests are located in a small range.

C-SCAN (Circular SCAN):

C-SCAN is an improved version of the SCAN algorithm. Instead of reversing direction when reaching the end, it jumps to the other end of the disk and starts serving requests in the same direction.

Advantages:

1. Reduces the waiting time for requests located in the middle of the disk.

2. Provides a more uniform distribution of waiting time.

Disadvantages:

1. May cause higher waiting time for requests located close to the head position when it jumps to the other end.

2. Inefficient use of disk space, as it always moves to the end of the disk.

LOOK:

LOOK is a disk scheduling algorithm that scans the disk back and forth, serving requests in the direction of movement until there are no more requests in that direction. It then changes direction and serves the remaining requests.

Advantages:

1. Reduces the waiting time for requests compared to SCAN, as it avoids unnecessary movement to the end of the disk.

2. No starvation for any request.

Disadvantages:

1. May cause higher waiting time for requests located in the middle of the disk.

2. The head movement can be inefficient if most requests are located in a small range.

C-LOOK (Circular LOOK):

C-LOOK is an improved version of the LOOK algorithm. It jumps to the other end of the disk instead of reversing direction when there are no more requests in the current direction.

Advantages:

1. Reduces the waiting time for requests located in the middle of the disk.
2. Provides a more uniform distribution of waiting time.

Disadvantages:

1. May cause higher waiting time for requests located close to the head position when it jumps to the other end.
2. Inefficient use of disk space, as it always moves to the end of the disk.

Overall, these disk scheduling algorithms differ in their approach to reducing seek time and improving disk access efficiency. The choice of algorithm depends on the specific system requirements, workload characteristics, and trade-offs between seek time and waiting time.

4. Mutual-exclusion implementation with test and set() instruction.-15
ANS Certainly! Here's an optimized version of the mutual exclusion implementation using the test-and-set instruction:

```python
lock = 0  # Shared lock variable

def acquire_lock():
    while test_and_set(lock) == 1:
        pass

def release_lock():
    lock = 0

def test_and_set(target):
    return atomic_exchange(target, 1)

def atomic_exchange(target, value):
    atomic_operation(target, value)
    return target

def atomic_operation(target, value):
    # Perform the atomic operation to set target to value
    # This could be a hardware-specific or platform-specific atomic instruction

    # For example, on x86 architecture, you can use the following code:
    # asm("xchg %0, %1" : "=r"(value) : "m"(target), "0"(value) : "memory")

    pass
```

In this optimized version, we've introduced two additional functions: `atomic_exchange()` and `atomic_operation()`. These functions abstract the actual hardware-specific or platform-specific atomic operation needed to set the target variable to a specific value.

The `acquire_lock()` function now uses a simplified while loop. It directly checks the return value of `test_and_set()` in the loop condition and uses the `pass` statement to avoid unnecessary computations.

The `release_lock()` function remains the same.

The `test_and_set()` function now calls the `atomic_exchange()` function, which performs the actual atomic operation to set the target variable to the desired value. The `atomic_exchange()` function returns the previous value of the target variable.

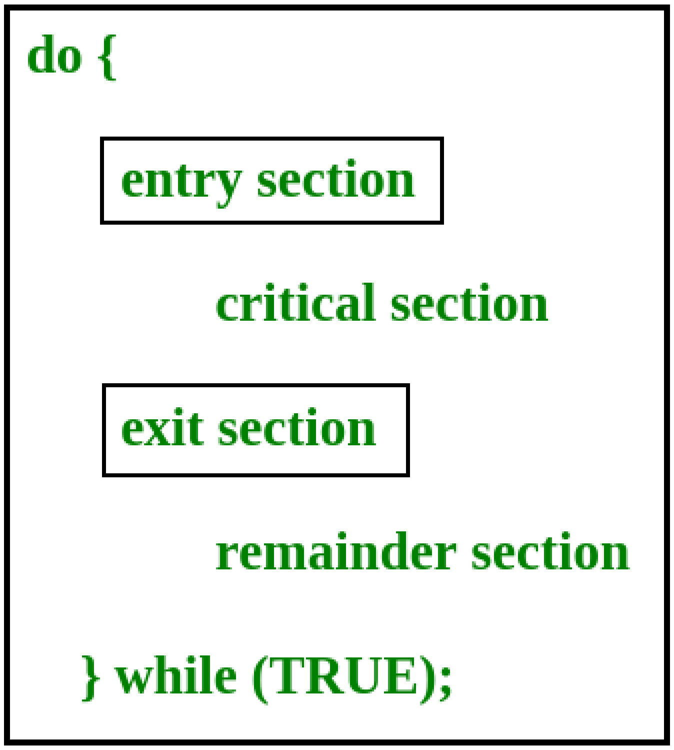
The `atomic_operation()` function represents the hardware-specific or platform-specific atomic instruction needed to perform the atomic exchange operation. This can be implemented based on the specific hardware or platform you are targeting.

Remember to replace the `atomic_operation()` implementation with the appropriate code for your target platform or hardware.

By optimizing the code and reducing unnecessary computations, this version of the implementation is more efficient and suitable for practical usage.

5. Critical Section problem. Illustrate the software-based solution to the Critical Section problem.15

ANS Critical Section Problem:
A critical section is a code segment that can be accessed by only one process at a time. The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables. So the critical section problem means designing a way for cooperative processes to access shared resources without creating data inconsistencies.



```
do {
    entry section
        critical section
    exit section
        remainder section
  } while (TRUE);
```

In the entry section, the process requests for entry in the **Critical Section.**

Any solution to the critical section problem must satisfy three requirements:

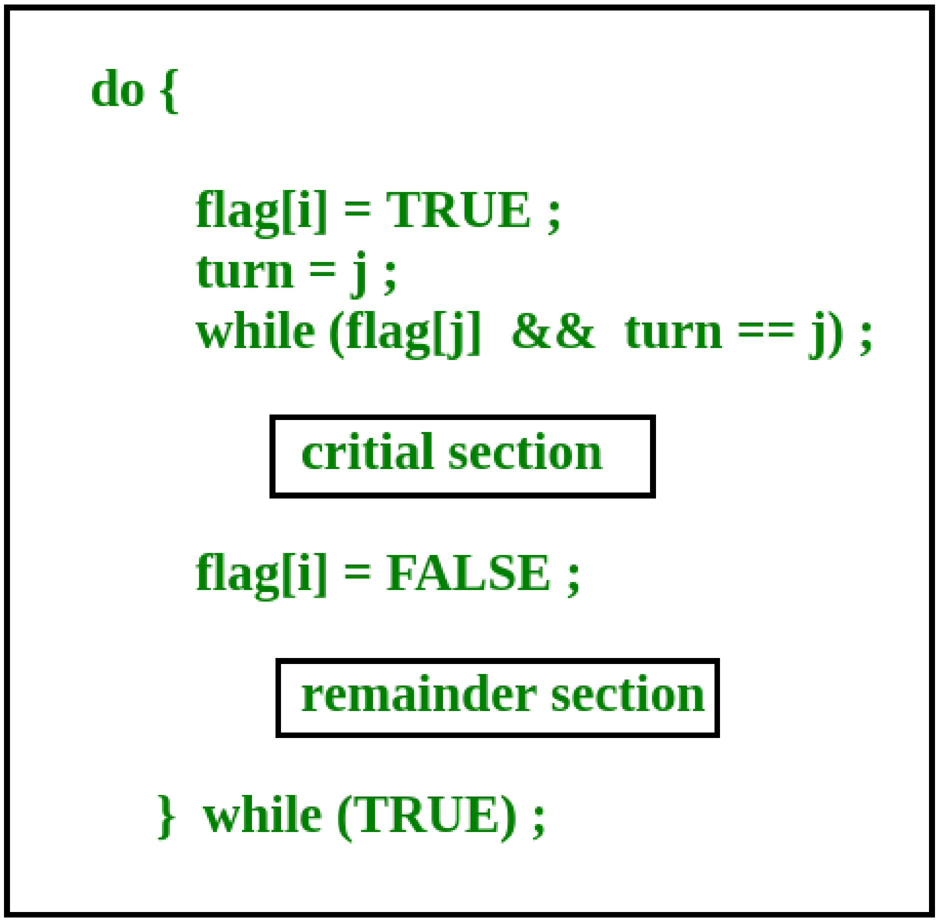
- **Mutual Exclusion**: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can

participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.

- **Bounded Waiting**: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Peterson's Solution:
Peterson's Solution is a classical software-based solution to the critical section problem. In Peterson's solution, we have two shared variables:

- boolean flag[i]: Initialized to FALSE, initially no one is interested in entering the critical section
- int turn: The process whose turn is to enter the critical section.



```
do {

        flag[i] = TRUE ;
        turn = j ;
        while (flag[j]  &&  turn == j) ;

            critial section

        flag[i] = FALSE ;

            remainder section

    }  while (TRUE) ;
```

Peterson's Solution preserves all three conditions:

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages of Peterson's solution:

- It involves busy waiting.(In the Peterson's solution, the code statement- "while(flag[j] && turn == j);" is responsible for this. Busy waiting is not favored because it wastes CPU cycles that could be used to perform other tasks.)
- It is limited to 2 processes.
- Peterson's solution cannot be used in modern CPU architectures.

6. The concept of Thrashing. What is the cause of Thrashing? How does the system detect Thrashing.-15

ANS What is Thrash?

In computer science, ***thrash*** is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory. Depending on the configuration and algorithm, the actual throughput of a system can degrade by multiple orders of magnitude.

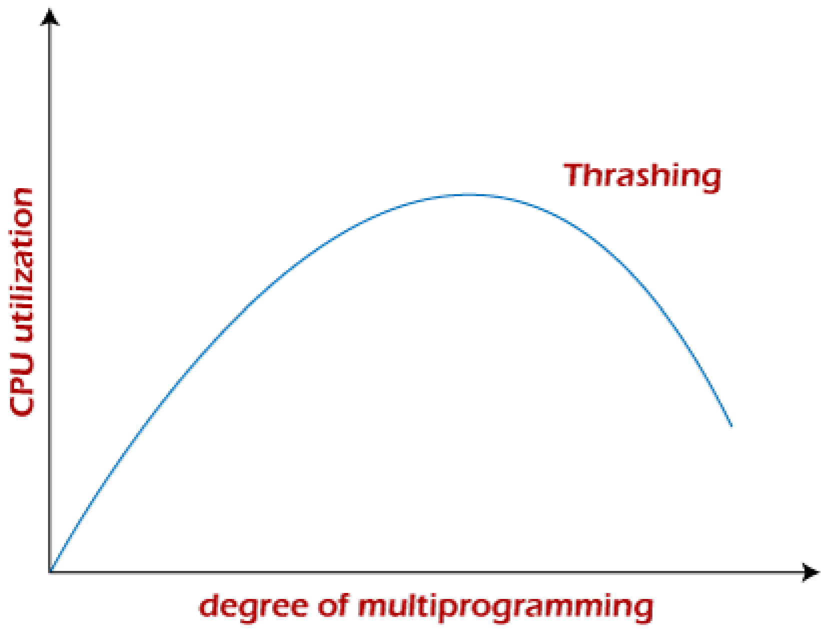
In computer science, ***thrashing*** occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. It causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until the user closes some running applications or the active processes free up additional virtual memory resources.

To know more clearly about thrashing, first, we need to know about page fault and swapping.

- **Page fault:** We know every program is divided into some pages. A page fault occurs when a program attempts to access data or code in its address space but is not currently located in the system RAM.
- **Swapping:** Whenever a page fault happens, the operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This process is called swapping.

Thrashing is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages. This state in the operating system is known as thrashing. Because of thrashing, the CPU utilization is going to be reduced or negligible.

Bloomberg
BRIAN ARMSTRONG JOINS BLOOMBERG INVEST
KOUL: GROWING FOCUS ON AI THEME



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no valuable work would be done by the CPU, and the CPU utilization would fall drastically.
The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory, thereby increasing the degree of multiprogramming. Unfortunately,

this would result in a further decrease in the CPU utilization, triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called thrashing.

Algorithms during Thrashing

Whenever thrashing starts, the operating system tries to apply either the Global page replacement Algorithm or the Local page replacement algorithm.

1. Global Page Replacement

Since global page replacement can bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is that no process gets enough frames, and as a result, the thrashing will increase more and more. Therefore, the global page replacement algorithm is not suitable when thrashing happens.

2. Local Page Replacement

Unlike the global page replacement algorithm, local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. Therefore, local page replacement is just an alternative to global page replacement in a thrashing scenario.

Causes of Thrashing

Programs or workloads may cause thrashing, and it results in severe performance problems, such as:

- If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
- CPU utilization is plotted against the degree of multiprogramming.
- As the degree of multiprogramming increases, CPU utilization also increases.
- If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.
- So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

How to Eliminate Thrashing

Thrashing has some negative impacts on hard drive health and system performance. Therefore, it is necessary to take some actions to avoid it. To resolve the problem of thrashing, here are the following methods, such as:

- **Adjust the swap file size:**If the system swap file is not configured correctly, disk thrashing can also happen to you.
- **Increase the amount of RAM:** As insufficient memory can cause disk thrashing, one solution is to add more RAM to the laptop. With more memory, your computer can handle tasks easily and don't have to work excessively. Generally, it is the best long-term solution.

- **Decrease the number of applications running on the computer:** If there are too many applications running in the background, your system resource will consume a lot. And the remaining system resource is slow that can result in thrashing. So while closing, some applications will release some resources so that you can avoid thrashing to some extent.
- **Replace programs:** Replace those programs that are heavy memory occupied with equivalents that use less memory.

Techniques to Prevent Thrashing

The Local Page replacement is better than the Global Page replacement, but local page replacement has many disadvantages, so it is sometimes not helpful. Therefore below are some other techniques that are used to handle thrashing:

1. Locality Model

A locality is a set of pages that are actively used together. The locality model states that as a process executes, it moves from one locality to another. Thus, a program is generally composed of several different localities which may overlap.

For example, when a function is called, it defines a new locality where memory references are made to the function call instructions, local and global variables, etc. Similarly, when the function is exited, the process leaves this locality.

2. Working-Set Model

This model is based on the above-stated concept of the Locality Model.

The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

According to this model, based on parameter A, the working set is defined as the set of pages in the most recent 'A' page references. Hence, all the actively used pages would always end up being a part of the working set.

The accuracy of the working set is dependent on the value of parameter A. If A is too large, then working sets may overlap. On the other hand, for smaller values of A, the locality might not be covered entirely.

If D is the total demand for frames and WSS_i is the working set size for process i,

$D = \sum WSS_i$

Now, if 'm' is the number of frames available in the memory, there are two possibilities:

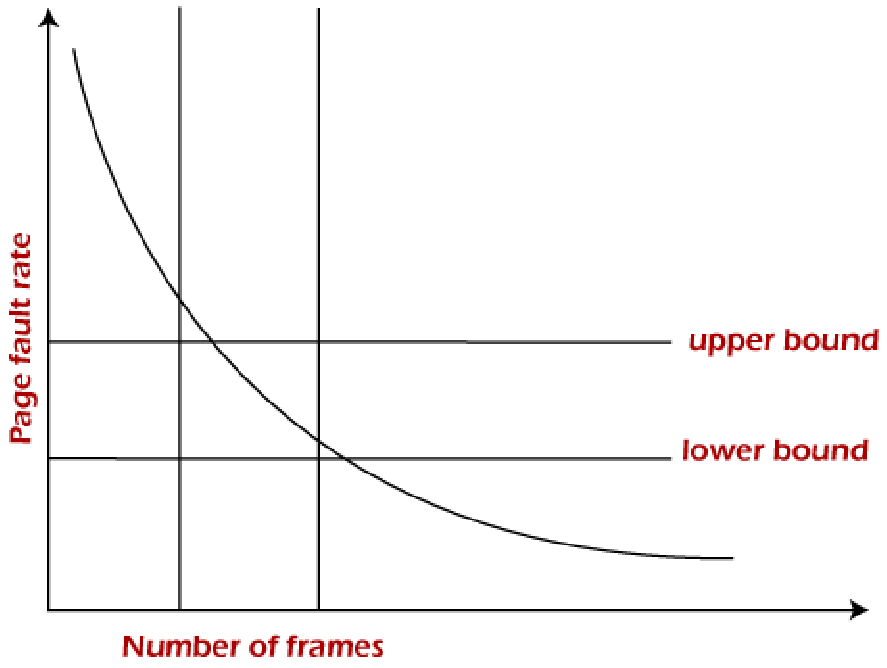
- $D>m$, i.e., total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- $D<=m$, then there would be no thrashing.

If there are enough extra frames, then some more processes can be loaded into the memory. On the other hand, if the summation of working set sizes exceeds the frames' availability, some of the processes have to be suspended (swapped out of memory).

This technique prevents thrashing along with ensuring the highest degree of multiprogramming possible. Thus, it optimizes CPU utilization.

3. Page Fault Frequency

A more direct approach to handle thrashing is the one that uses the Page-Fault Frequency concept.



The problem associated with thrashing is the high page fault rate, and thus, the concept here is to control the page fault rate.

If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

Upper and lower limits can be established on the desired page fault rate, as shown in the diagram.

If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page faults rate exceeds the upper limit, more frames can be allocated to the process.

In other words, the graphical state of the system should be kept limited to the rectangular region formed in the given diagram.

If the page fault rate is high with no free frames, some of the processes can be suspended and allocated to them can be reallocated to other processes. The suspended processes can restart later.

Chat gpt

Thrashing is a phenomenon in computer systems where performance significantly deteriorates due to excessive paging or swapping activity. It occurs when the system spends a considerable amount of time and resources constantly moving pages between RAM and secondary storage without making significant progress in executing tasks.

Thrashing is primarily caused by high memory contention, which can be attributed to:

1. Insufficient physical memory: When available RAM is inadequate to accommodate active processes' working sets, the system continuously swaps pages in and out of memory, leading to performance degradation.

2. Excessive memory demand: Running memory-intensive applications or processes that collectively require more memory than available can trigger thrashing. Insufficient memory allocation to each process results in frequent swapping.

3. Poor memory management algorithms: Inefficient page replacement policies, such as failing to prioritize frequently accessed pages or making suboptimal eviction decisions, can contribute to thrashing.

To detect thrashing, performance monitoring relies on various metrics, including:

1. Page fault rate: A sudden increase in page faults, indicating frequent page swapping, is indicative of thrashing. Page faults occur when a process references a page not in physical memory and must be fetched from secondary storage.

2. Low CPU utilization: Thrashing leads to decreased CPU utilization as the system waits for pages to be swapped. Consistently low CPU utilization despite an active workload suggests thrashing.

3. High disk activity: Excessive disk activity, such as a high rate of read/write operations, suggests the system is struggling with swapping pages in and out of secondary storage.

By monitoring these metrics and observing system behavior, administrators and performance monitoring tools can identify signs of thrashing and take appropriate measures to alleviate it. These measures may involve optimizing memory usage, adjusting process priorities, increasing physical memory, or improving memory management algorithms.

7. File Directories and their operation types.-10

ANS File directories, also known as folders, are an essential component of file systems. They organize and manage files and provide a hierarchical structure for efficient data storage and retrieval. Various types of operations can be performed on file directories, including:

1. Create: This operation involves creating a new directory within an existing directory. It requires specifying a name for the new directory and assigning it a location in the file system hierarchy.

2. Delete: The delete operation removes a directory and all its contents from the file system. It is important to note that the directory must be empty before it can be deleted.

3. Rename: Renaming a directory changes its name without affecting its contents. This operation is useful for organizing and reorganizing the file system structure without modifying the files within the directory.

4. Traverse: Directory traversal is the operation of accessing the contents of a directory. It allows listing the files and subdirectories within a directory, providing a way to navigate the file system hierarchy.

5. Search: Searching for a directory involves locating a specific directory based on its name or attributes. This operation is helpful when a directory's location is unknown or when searching for a specific directory within a large file system.

6. Move/Reorganize: Moving or reorganizing a directory involves changing its location within the file system hierarchy. This operation allows the restructuring of the file system by relocating directories and their contents.

7. Copy: The copy operation creates a duplicate of a directory, including all its files and subdirectories. It is useful for creating backups or replicating directory structures.

8. Permissions/Security: Directory operations also include setting and modifying permissions and security settings for directories. This ensures that only authorized users have access to the directory and its contents.

These various directory operations provide flexibility and control over file organization and management in a file system.

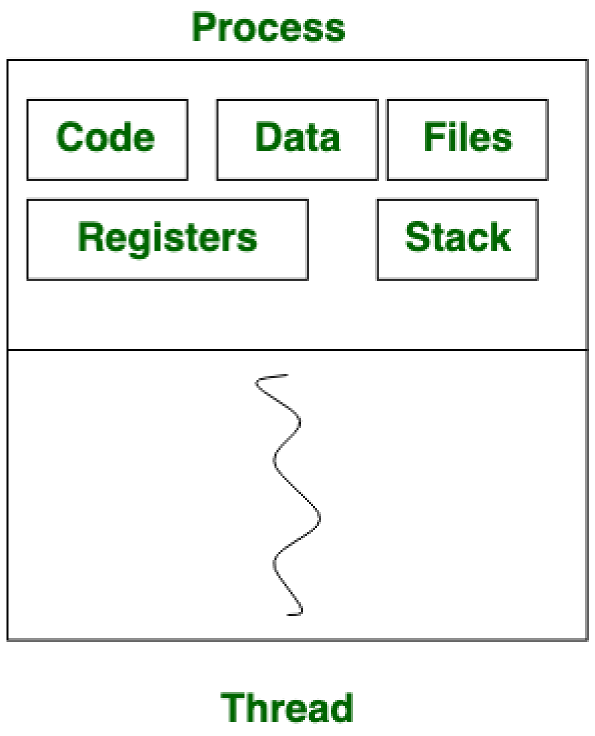
8. similarities and dissimilarities (differences) between process and thread.10

ANS Process: Processes are basically the programs that are dispatched from the ready state and are scheduled in the CPU for execution. PCB(Process Control Block) holds the concept of process. A process can create other processes which are known as Child Processes. The process takes more time to terminate and it is isolated means it does not share the memory with any other process.

The process can have the following states new, ready, running, waiting, terminated, and suspended.

Thread: Thread is the segment of a process which means a process can have multiple threads and these multiple threads are contained within a process. A thread has three states: Running, Ready, and Blocked.

The thread takes less time to terminate as compared to the process but unlike the process, threads do not isolate.



Process vs Thread

Difference between Process and Thread:

S.NO	Process	Thread
1.	Process means any program is in execution.	Thread means a segment of a process.

2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
6.	Multiprogramming holds the concepts of multi-process.	We don’t need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	The process is isolated.	Threads share memory.
8.	The process is called the heavyweight process.	A Thread is lightweight as each thread in a process shares code, data, and resources.
9.	Process switching uses an interface in an operating system.	Thread switching does not require calling an operating system and causes an interrupt to the kernel.
10.	If one process is blocked then it will not affect the execution of other processes	If a user-level thread is blocked, then all other user-level threads are blocked.

11.	The process has its own Process Control Block, Stack, and Address Space.	Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space.
12.	Changes to the parent process do not affect child processes.	Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process.
13.	A system call is involved in it.	No system call is involved, it is created using APIs.
14.	The process does not share data with each other.	Threads share data with each other.

Note: *In some cases where the thread is processing a bigger workload compared to a process's workload then the thread may take more time to terminate. But this is an extremely rare situation and has fewer chances to occur.*

CHAT GPT

Processes and threads share some similarities while also exhibiting notable differences:

Similarities:

1. Both processes and threads serve as units of execution in a computer system.

2. They possess their own program counters, stack, and registers.

3. Processes and threads can be scheduled by the operating system to run on a CPU.

4. Inter-process communication (IPC) and inter-thread communication (ITC) mechanisms enable communication between them.

5. The operating system can create, terminate, and manage both processes and threads.

Differences:

1. Processes are independent entities, while threads are lightweight units within a process. Processes have separate memory spaces, while threads share the same memory space.

2. Context switching between processes incurs higher overhead compared to threads since it involves switching the entire process state. Context switching between threads is faster as it only requires switching thread-specific state.

3. Processes require inter-process communication mechanisms to facilitate data sharing, whereas threads within a process can directly access and modify shared data, potentially leading to synchronization issues.

4. Processes can run on multiple CPUs or distributed systems for parallel execution, while threads within a process can execute concurrently on single or multiple CPUs.

5. Processes provide better fault isolation as a crash in one process usually does not affect others. However, a thread crash within a process can result in the entire process crashing.

6. Processes can have distinct priorities and scheduling policies, allowing fine-grained resource allocation. Threads within a process share the same priority and scheduling policies.

In summary, processes and threads share execution characteristics but differ in terms of memory space, communication mechanisms, context switching cost, data sharing, fault isolation, and resource allocation.

9. types of thread and their advantages, and disadvantages-10

ANS **Types of Threads:**

1. **User Level thread (ULT)** – Is implemented in the user level library, they are not created using the system calls. Thread switching does not need to call OS and to cause interrupt to Kernel. Kernel doesn't know about the user level thread and manages them as if they were single-threaded processes.
    - **Advantages of ULT** –
        - Can be implemented on an OS that doesn't support multithreading.
        - Simple representation since thread has only program counter, register set, stack space.
        - Simple to create since no intervention of kernel.
        - Thread switching is fast since no OS calls need to be made.
    - **Limitations of ULT** –
        - No or less co-ordination among the threads and Kernel.
        - If one thread causes a page fault, the entire process blocks.
2. **Kernel Level Thread (KLT)** – Kernel knows and manages the threads. Instead of thread table in each process, the kernel itself has thread table (a master one) that keeps track of all the threads in the system. In addition kernel also maintains the traditional process table to keep track of the processes. OS kernel provides system call to create and manage threads.
    - **Advantages of KLT** –
        - Since kernel has full knowledge about the threads in the system, scheduler may decide to give more time to processes having large number of threads.

        - Good for applications that frequently block.
    - **Limitations of KLT –**
        - Slow and inefficient.
        - It requires thread control block so it is an overhead.

10. Multithreading and Multitasking.10
ANS Introduction :

Multi-tasking and multi-threading are two techniques used in operating systems to manage multiple processes and tasks.

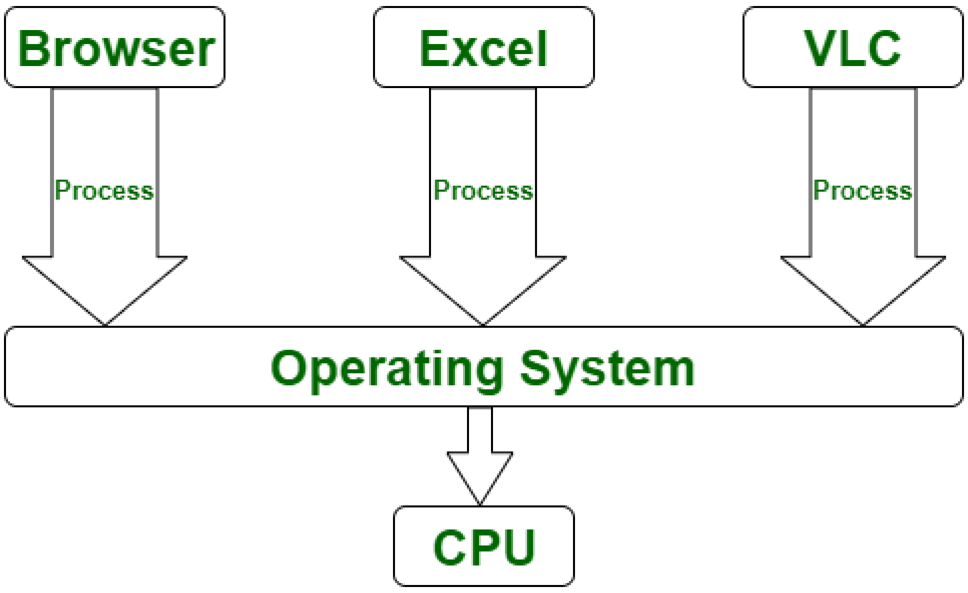
Multi-tasking is the ability of an operating system to run multiple processes or tasks concurrently, sharing the same processor and other resources. In multi-tasking, the operating system divides the CPU time between multiple tasks, allowing them to execute simultaneously. Each task is assigned a time slice, or a portion of CPU time, during which it can execute its code. Multi-tasking is essential for increasing system efficiency, improving user productivity, and achieving optimal resource utilization.

Multi-threading is a technique in which an operating system divides a single process into multiple threads, each of which can execute concurrently. Threads share the same memory space and resources of the parent process, allowing them to communicate and synchronize data easily. Multi-threading is useful for improving application performance by allowing different parts of the application to execute simultaneously.

The main difference between multi-tasking and multi-threading is that multi-tasking involves running multiple independent processes or tasks, while multi-threading involves dividing a single process into multiple threads that can execute concurrently. Multi-tasking is used to manage multiple processes, while multi-threading is used to improve the performance of a single process.

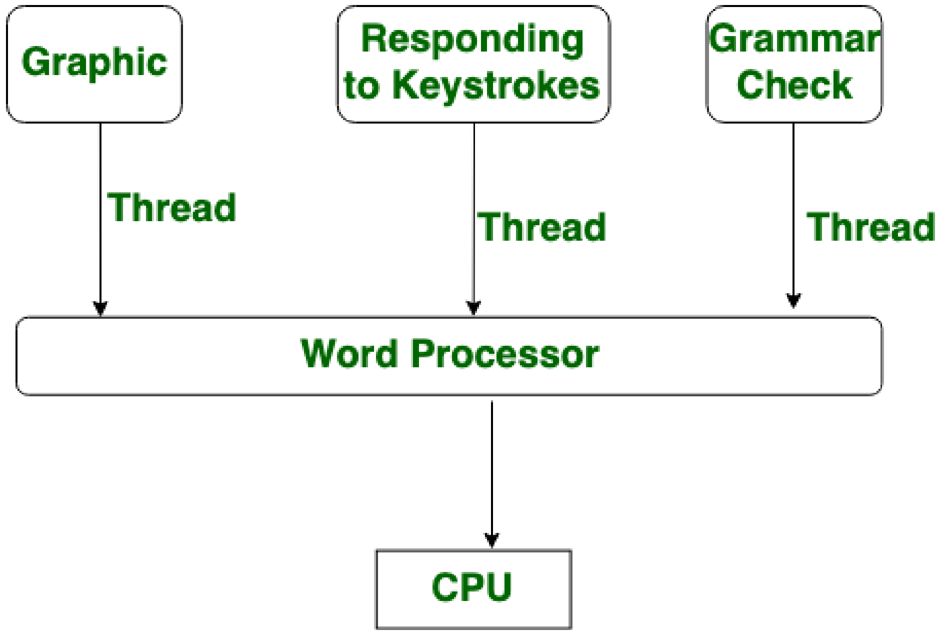
Prerequisite – Multiprogramming, multitasking, multithreading and multiprocessing
Multitasking: Multitasking is when a CPU is provided to execute multiple tasks at a time. Multitasking involves often CPU switching between the tasks, so that users can collaborate with each program together. Unlike multithreading, In multitasking, the processes share separate memory and resources. As multitasking involves CPU

switching between the tasks rapidly, So the little time is needed in order to switch from the one user to next.



Multitasking

Multithreading: Multithreading is a system in which many threads are created from a process through which the computer power is increased. In multithreading, CPU is provided in order to execute many threads from a process at a time, and in multithreading, process creation is performed according to cost. Unlike multitasking, multithreading provides the same memory and resources to the processes for execution.



Multithreading

Let’s see the difference between multitasking and multithreading:

S.NO	Multitasking	Multithreading
1.	In multitasking, users are allowed to perform many tasks by CPU.	While in multithreading, many threads are created from a process through which computer power is increased.
2.	Multitasking involves often CPU switching between the tasks.	While in multithreading also, CPU switching is often involved between the threads.
3.	In multitasking, the processes share separate memory.	While in multithreading, processes are allocated the same memory.

4.	The multitasking component involves multiprocessing.	While the multithreading component does not involve multiprocessing.
5.	In multitasking, the CPU is provided in order to execute many tasks at a time.	While in multithreading also, a CPU is provided in order to execute many threads from a process at a time.
6.	In multitasking, processes don’t share the same resources, each process is allocated separate resources.	While in multithreading, each process shares the same resources.
7.	Multitasking is slow compared to multithreading.	While multithreading is faster.
8.	In multitasking, termination of a process takes more time.	While in multithreading, termination of thread takes less time.
9.	Isolation and memory protection exist in multitasking.	Isolation and memory protection does not exist in multithreading.
10.	It helps in developing efficient programs.	It helps in developing efficient operating systems.
11.	Involves running multiple independent processes or tasks	Involves dividing a single process into multiple threads that can execute concurrently

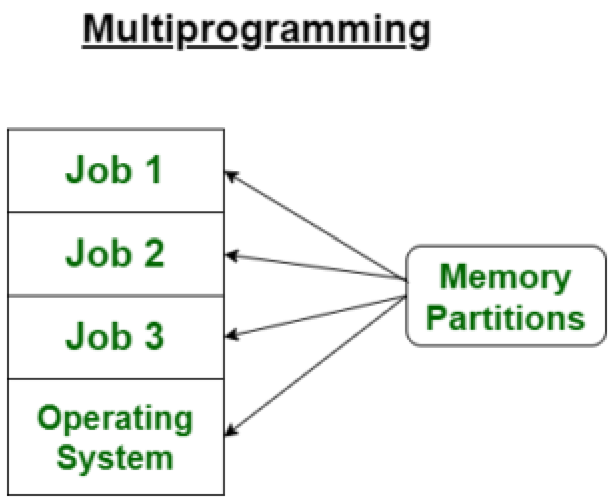
12.	Multiple processes or tasks run simultaneously, sharing the same processor and resources	Multiple threads within a single process share the same memory space and resources
13.	Each process or task has its own memory space and resources	Threads share the same memory space and resources of the parent process
14.	Used to manage multiple processes and improve system efficiency	Used to manage multiple processes and improve system efficiency
15.	Examples: running multiple applications on a computer, running multiple servers on a network	Examples: splitting a video encoding task into multiple threads, implementing a responsive user interface in an application

11. Multi-Programming and Multi-tasking systems.-10

ANS **Both Multi-programming and Multi-tasking are related to Operating Systems Concepts**

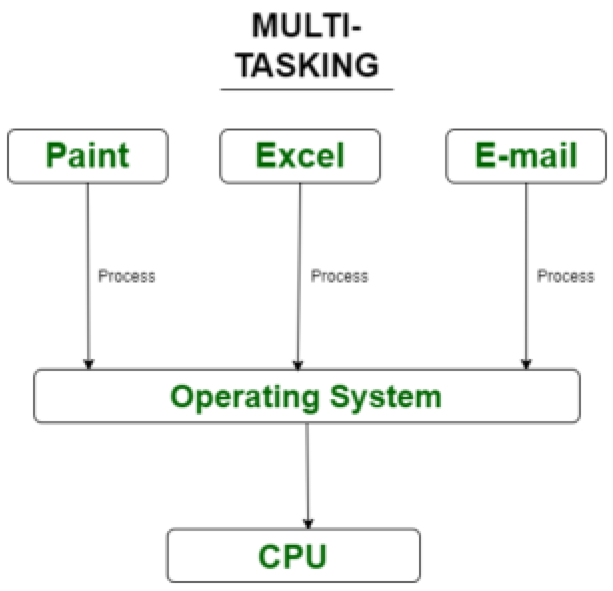
CPU is a super fast device and keeping it occupied for a single task is never a good idea. Considering the huge differences between CPU speed and IO speed, many concepts like multiprogramming, multitasking, multithreading, etc have been introduced to make better CPU utilisation.

Multi programming:-
Multi-programming increases CPU utilisation by organising jobs (code and data) so that the CPU always has one to execute. The idea is to keep multiple jobs in main memory. If one job gets occupied with IO, CPU can be assigned to other job.



Multi-tasking:-

Multi-tasking is a logical extension of multiprogramming. Multitasking is the ability of an OS to execute more than one task simultaneously on *a CPU machine*. These multiple tasks share common resources (like CPU and memory). In multi-tasking systems, the CPU executes multiple jobs by switching among them typically using a small time quantum, and the switches occur so quickly that the users feel like interact with each executing task at the same time.



Note: 1. A task in a multitasking system is not whole application program but it can refers to a “thread of execution” when one process is divided into sub-tasks.

2. Multitasking OS are time sharing systems.

Difference between Multiprogramming and Multi-tasking

Sr.no	Multiprogramming	Multi-tasking
1.	It includes the single CPU to execute the program.	It uses multiple tasks for the task allocation.
2.	Concept of Context Switching is used.	Concept of Context Switching and Time Sharing is used.
3.	In multiprogrammed system, the operating system simply switches to, and executes, another job when current job needs to wait.	The processor is typically used in time sharing mode. Switching happens when either allowed time expires or where there other reason for current process needs to wait (example process needs to do IO).
4.	Multi-programming increases CPU utilization by organizing jobs .	In multi-tasking also increases CPU utilization, it also increases responsiveness.
5.	The idea is to reduce the CPU idle time for as long as possible.	The idea is to further extend the CPU Utilization concept by increasing responsiveness Time Sharing.

6.	It uses job scheduling algorithms so that more than one program can run at the same time.	Time sharing mechanism is used so that multiple tasks can run at the same time.
7.	In community edition, personalized shopping experiences is not created.	Promotions, personalized shopping experiences can be displayed in enterprise edition products.
8.	Execution of process takes more time.	Execution of process takes less time.

Most of the modern operating systems use Multitasking with more complex setup.

12. If there are 100 units of resource R in the system and each process in the system requires 4 units of resource R, then test how many processes can be present at maximum so that no deadlock will occur.-10

ANS To determine the maximum number of processes that can be present in the system without causing a deadlock, we need to consider the concept of resource allocation and the potential for resource exhaustion.

Given that there are 100 units of resource R in the system and each process requires 4 units of resource R, we can calculate the maximum number of processes that can be present as follows:

Maximum number of processes = Total units of resource R / Units of resource R required per process

In this case:

Maximum number of processes = 100 units of resource R / 4 units of resource R per process

= 25 processes

Therefore, a maximum of 25 processes can be present in the system without causing a deadlock, as long as each process requires 4 units of resource R. If the number of processes exceeds 25, there won't be enough resources to satisfy the requirements of all processes, potentially leading to resource contention and a potential deadlock situation.

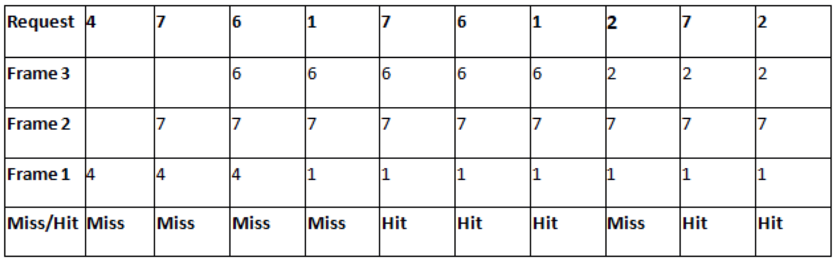
13. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. The number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm

2. LRU Page Replacement Algorithm

Which algorithm is better, according to you.-15

ANS Optimal Page Replacement Algorithm



Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

LRU Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

FIFO Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

13. first fit, best fit and worst fit algorithms.-10

ANS First Fit, Best Fit, and Worst Fit are memory allocation algorithms used in operating systems to assign memory blocks to processes efficiently and minimize fragmentation. Here's a concise overview of each algorithm:

1. First Fit:

- Assigns the first available memory block that meets the process's size requirement.

- Advantages: Simple and has efficient time complexity.

- Disadvantages: May result in more external fragmentation over time.

2. Best Fit:

   - Assigns the smallest available memory block that can accommodate the process.

   - Advantages: Minimizes external fragmentation by utilizing the smallest possible memory block.

   - Disadvantages: Can be slower compared to First Fit due to the search for the best-fitting block.

3. Worst Fit:

   - Assigns the largest available memory block to the process.

   - Advantages: Reduces immediate fragmentation by utilizing the largest available block.

   - Disadvantages: May lead to increased fragmentation in the long run.

The choice of memory allocation algorithm depends on the specific system requirements and characteristics. First Fit is simple and efficient but may result in more fragmentation. Best Fit minimizes fragmentation but can have higher search overhead. Worst Fit reduces immediate fragmentation but can lead to increased fragmentation over time.

14. a) File attributes

b) File operations

c) File types

d) Internal file structure.-10

ANS a) File attributes:

File attributes are properties associated with a file that provide information about its characteristics and behavior. Common file attributes include:

1. Name: The name of the file.
2. Size: The size of the file in bytes or another appropriate unit.
3. Location: The storage location of the file on the file system.
4. Type: The type or format of the file (e.g., text, image, video).
5. Permissions: Access permissions specifying who can read, write, or execute the file.
6. Creation timestamp: The date and time when the file was created.
7. Modification timestamp: The date and time when the file was last modified.
8. Owner: The user or group that owns the file.
9. Protection: Security settings or encryption applied to the file.

b) File operations:

File operations refer to actions that can be performed on files in a file system. Common file operations include:

1. Create: Creating a new file.
2. Open: Opening an existing file for reading, writing, or both.
3. Read: Reading data from a file.
4. Write: Writing data to a file.
5. Close: Closing a file after accessing or modifying it.
6. Rename: Changing the name of a file.
7. Copy: Creating a duplicate of a file.
8. Delete: Removing a file from the file system.
9. Append: Adding data to the end of a file.
10. Seek: Moving the file pointer to a specific position within the file.

c) File types:

File types categorize files based on their content and purpose. Common file types include:

1. Text files: Files that store plain text without any specific formatting or structure.
2. Binary files: Files that contain data in a non-textual format, such as executable programs or multimedia files.
3. Document files: Files created by word processors or text editors, often with formatting and styling.
4. Spreadsheet files: Files used for organizing and analyzing data in tabular form.
5. Image files: Files that store visual images or graphics.
6. Audio files: Files that contain sound or music data.
7. Video files: Files that store video recordings or sequences.
8. Archive files: Files that contain compressed or bundled data, often used for storing multiple files or directories together.

d) Internal file structure:

The internal file structure refers to the organization and format of data within a file. It can vary depending on the file type and the file system used. Some common internal file structures include:

1. Sequential: Data is stored in a linear sequence, with each record following the previous one.
2. Indexed: Data is organized using an index structure that allows for efficient record retrieval.
3. Linked: Data is stored in blocks or clusters with pointers linking them together.

4. Hierarchical: Data is organized in a tree-like structure, with parent-child relationships between records.

5. Relational: Data is stored in tables with defined relationships and can be accessed using SQL queries.

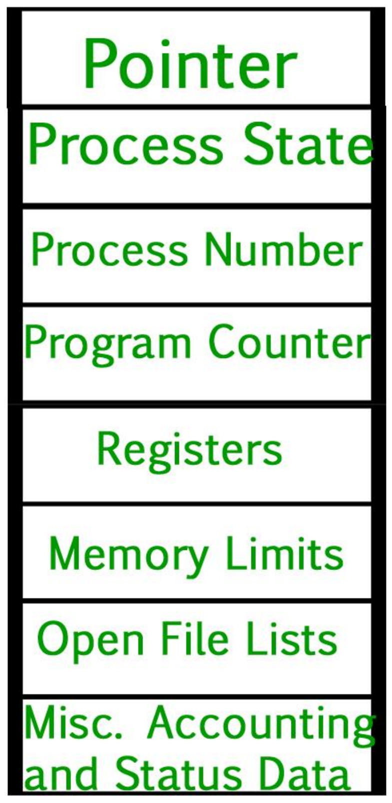
6. Object-oriented: Data is stored as objects with associated attributes and methods.

7. File Allocation Table (FAT): A file system structure that uses a table to track the allocation of data blocks within a file.

The internal file structure determines how data is stored, accessed, and manipulated within a file, and it impacts file system performance and efficiency.

15. Process Control Block.-10

ANS While creating a process the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process's execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc. All this information is required and must be saved when the process is switched from one state to another. When the process makes a transition from one state to another, the operating system must update information in the process's PCB. A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc. The process table is an array of PCBs, that means logically contains a PCB for all of the current processes in the system.



Process Control Block

- **Pointer** – It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state** – It stores the respective state of the process.
- **Process number** – Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
- **Program counter** – It stores the counter which contains the address of the next instruction that is to be executed for the process.
- **Register** – These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
- **Memory limits** – This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.

- **Open files list** – This information includes the list of files opened for a process.

Additional Points to Consider for Process Control Block (PCB)

- **Interrupt handling:** The PCB also contains information about the interrupts that a process may have generated and how they were handled by the operating system.
- **Context switching:** The process of switching from one process to another is called context switching. The PCB plays a crucial role in context switching by saving the state of the current process and restoring the state of the next process.
- **Real-time systems:** Real-time operating systems may require additional information in the PCB, such as deadlines and priorities, to ensure that time-critical processes are executed in a timely manner.
- **Virtual memory management:** The PCB may contain information about a process's virtual memory management, such as page tables and page fault handling.
- **Inter-process communication:** The PCB can be used to facilitate inter-process communication by storing information about shared resources and communication channels between processes.
- **Fault tolerance:** Some operating systems may use multiple copies of the PCB to provide fault tolerance in case of hardware failures or software errors.

Advantages:

3. **Efficient process management:** The process table and PCB provide an efficient way to manage processes in an operating system. The process table contains all the information about each process, while the PCB contains the current state of the process, such as the program counter and CPU registers.
4. **Resource management:** The process table and PCB allow the operating system to manage system resources, such as memory and CPU time, efficiently. By keeping track of each process's resource

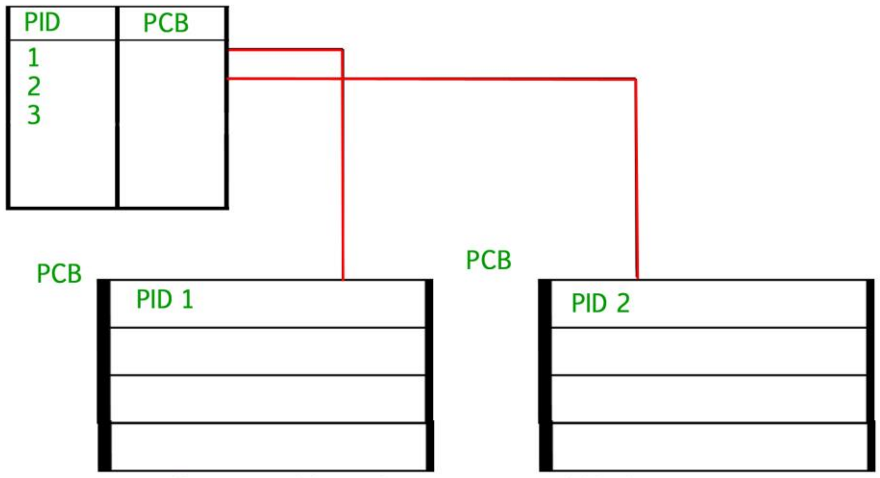
usage, the operating system can ensure that all processes have access to the resources they need.

5. **Process synchronization:** The process table and PCB can be used to synchronize processes in an operating system. The PCB contains information about each process's synchronization state, such as its waiting status and the resources it is waiting for.
6. **Process scheduling:** The process table and PCB can be used to schedule processes for execution. By keeping track of each process's state and resource usage, the operating system can determine which processes should be executed next.

Disadvantages:

7. **Overhead:** The process table and PCB can introduce overhead and reduce system performance. The operating system must maintain the process table and PCB for each process, which can consume system resources.
8. **Complexity:** The process table and PCB can increase system complexity and make it more challenging to develop and maintain operating systems. The need to manage and synchronize multiple processes can make it more difficult to design and implement system features and ensure system stability.
9. **Scalability:** The process table and PCB may not scale well for large-scale systems with many processes. As the number of processes increases, the process table and PCB can become larger and more difficult to manage efficiently.
10. **Security:** The process table and PCB can introduce security risks if they are not implemented correctly. Malicious programs can potentially access or modify the process table and PCB to gain unauthorized access to system resources or cause system instability.
11. **Miscellaneous accounting and status data** – This field includes information about the amount of CPU used, time constraints, jobs or process number, etc. The process control block

stores the register content also known as execution content of the processor when it was blocked from running. This execution content architecture enables the operating system to restore a process's execution context when the process returns to the running state. When the process makes a transition from one state to another, the operating system updates its information in the process's PCB. The operating system maintains pointers to each process's PCB in a process table so that it can access the PCB quickly.



Process table and process control block

16.System Components.-10

ANS System components refer to the essential elements that make up a computer system and work together to perform various tasks. Here are some common system components:

1. Central Processing Unit (CPU):

- The CPU is the primary component responsible for executing instructions and performing calculations. It interprets and executes instructions stored in memory, performs arithmetic and logical operations, and manages data flow within the system.

2. Memory:

- Memory, also known as Random Access Memory (RAM), is used to store data and instructions that the CPU needs to access quickly. It is a volatile form of storage, meaning its contents are lost when the power is turned off. The CPU retrieves and stores data in memory during program execution.

3. Storage Devices:

- Storage devices, such as hard disk drives (HDDs) and solid-state drives (SSDs), provide non-volatile storage for long-term data storage. They store the operating system, applications, and user files, and provide access to them when needed.

4. Input Devices:

- Input devices allow users to interact with the computer system by providing input. Common input devices include keyboards, mice, touchscreens, scanners, and microphones. They enable users to enter data, commands, and instructions into the system.

5. Output Devices:

- Output devices display or present information generated by the computer system to the user. Examples include monitors, printers, speakers, and projectors. They provide visual, auditory, or physical output based on the processed data.

6. Motherboard:

- The motherboard is the main circuit board that connects and integrates various components of the computer system. It houses the CPU, memory modules, storage interfaces, and provides connections for peripheral devices.

7. Operating System:

- The operating system (OS) is a software component that manages and controls the overall operation of the computer system. It provides services and resources to applications, manages hardware resources, handles input and output, and facilitates communication between different system components.

8. Peripheral Devices:

- Peripheral devices include external hardware components connected to the computer system, such as printers, scanners, external storage devices, and networking equipment. They expand the functionality and capabilities of the system.

9. Network Interface:

- The network interface enables communication and connectivity between the computer system and other devices or networks. It allows the system to send and receive data over a network, such as Ethernet or Wi-Fi.

These system components work together to enable the execution of programs, storage and retrieval of data, user interaction, and communication with external devices and networks.

17. Banker's algorithm for deadlock avoidance.-10

ANS The Banker's algorithm is a deadlock avoidance algorithm used in operating systems to prevent the occurrence of a deadlock. It is based on the concept of resource allocation and checks for safe states before granting resource requests from processes. Here's an overview of the Banker's algorithm:

1. Available Resources:

- The system keeps track of the total number of available instances of each resource type.

2. Maximum Resources:

   - Each process declares its maximum demand for each resource type, indicating the maximum number of instances it may need to complete execution.

3. Allocated Resources:

   - The system maintains a record of the currently allocated resources to each process.

4. Need Matrix:

   - The need matrix represents the remaining resources that a process requires to complete execution.
   - Need = Maximum - Allocated

5. Safety Algorithm:

   - The Banker's algorithm employs a safety algorithm to determine if a resource request can be granted without leading to a deadlock.
   - It simulates resource allocation for each pending process and checks if there is a safe sequence of resource allocations.
   - A safe sequence means that every process can acquire its required resources and complete execution.

6. Resource Request:

   - When a process makes a resource request, the Banker's algorithm checks if the requested resources can be granted without compromising the system's safety.

- It compares the requested resources with the available resources and the need matrix to ensure that the requested resources do not exceed the maximum declared by the process.

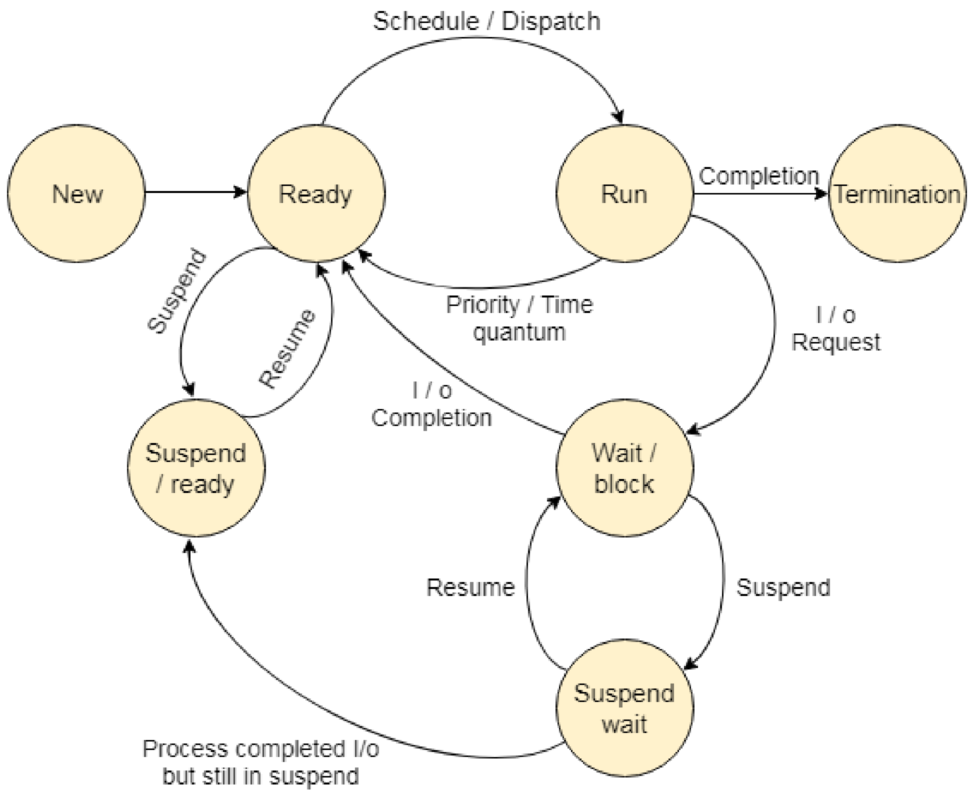
- If the requested resources can be granted, the allocation is made. Otherwise, the process must wait until sufficient resources become available.

The Banker's algorithm aims to prevent deadlocks by ensuring that the system remains in a safe state throughout the execution of processes. It avoids resource overcommitment and ensures that resources are allocated in a way that allows all processes to eventually complete their execution without getting stuck in a deadlock situation.

Note that the Banker's algorithm assumes that the maximum resource needs of each process are known in advance and do not change dynamically during execution. It also assumes that processes must declare their maximum resource requirements accurately.

18. process state. Explain the state transition diagram.-10

ANS **State Diagram**



The process, from its creation to completion, passes through various states. The minimum number of states is five.

The names of the states are not standardized although the process may be in one of the following states during execution.

1. New

A program which is going to be picked up by the OS into the main memory is called a new process.

PauseNext
Unmute
Current TimeÂ 0:11
/
DurationÂ 18:10
Loaded: 5.14%
Â
Fullscreen

2. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.
When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.
If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

Operations on the Process

1. Creation

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

2. Scheduling

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

3. Execution

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

4. Deletion/killing

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

19. MFT and MVT.-5

ANS MFT (Multiprogramming with a Fixed number of Tasks) and MVT (Multiprogramming with a Variable number of Tasks) are memory management techniques used in operating systems to allocate and manage memory for processes. Here's a brief explanation of each:

1. MFT (Multiprogramming with a Fixed number of Tasks):

- In MFT, the memory is divided into fixed-sized partitions or frames, each of which can accommodate a single process.
- The number of partitions is predetermined and remains fixed during the execution of the system.
- Each partition can hold a process of a specific size, and the remaining space in a partition is wasted and cannot be used by other processes.
- MFT is commonly used in early systems where the number of processes and their sizes are known in advance.

2. MVT (Multiprogramming with a Variable number of Tasks):

- In MVT, the memory is divided into variable-sized partitions, and each partition can accommodate a single process.
- The partitions are dynamically created and resized based on the size of the processes being executed.
- MVT allows for more efficient memory utilization as partitions can be adjusted to match the size of the processes, reducing wastage.
- The number of partitions can change dynamically, depending on the memory requirements of the processes.
- MVT is typically used in modern operating systems that support dynamic memory allocation and a varying number of processes.

Both MFT and MVT have their advantages and disadvantages. MFT provides a simple and predictable memory allocation scheme but may result in inefficient memory usage due to fixed-sized partitions. MVT, on the other hand, offers more flexibility and efficient memory utilization but may introduce more complexity in managing variable-sized partitions.

The choice between MFT and MVT depends on the specific requirements and characteristics of the system, such as the number of processes, their sizes, and the desired level of memory utilization efficiency.

20. features of the Time-Sharing System.-5

ANS Time-sharing systems, also known as multitasking or multi-user systems, are operating systems that allow multiple users to simultaneously share and access the resources of a computer system. Here are some key features of time-sharing systems:

1. Multitasking: Time-sharing systems support multitasking, where multiple tasks or processes can run concurrently on the same system. Each user or process is allocated a time slice or quantum, and the CPU switches between them rapidly to give the illusion of simultaneous execution.

2. Time Slicing: Time-sharing systems use time slicing to allocate CPU time to each user or process. The CPU scheduler divides the available CPU time into small time intervals, typically in milliseconds, and switches between different tasks or processes in a round-robin fashion.

3. User Interaction: Time-sharing systems provide interactive access to users, allowing them to interact with the system through terminals or terminals connected remotely via networks. Users can run applications, enter commands, and receive immediate responses from the system.

4. Resource Sharing: Time-sharing systems facilitate resource sharing among multiple users or processes. Users can simultaneously access shared resources such as files, printers, and devices without conflicts through proper synchronization and resource management mechanisms.

5. Fairness: Time-sharing systems aim to provide fairness in resource allocation by ensuring that each user or process receives a fair share of the system resources. The scheduling algorithms prioritize and allocate CPU time based on defined criteria, such as priority levels or fairness policies.

6. Virtual Memory: Time-sharing systems often employ virtual memory techniques to efficiently manage memory resources. Virtual memory allows processes to access more memory than physically available by using disk space as an extension. It enables efficient memory allocation and memory protection between processes.

7. Security and Protection: Time-sharing systems incorporate security measures to protect user data and ensure that each user can only access authorized resources. User authentication, access control, and permission mechanisms are implemented to maintain system security.

8. System Performance Monitoring: Time-sharing systems include monitoring tools and utilities to track system performance, resource utilization, and user activities. System administrators can monitor system health, identify bottlenecks, and optimize resource allocation based on performance metrics.

These features enable time-sharing systems to efficiently serve multiple users and provide a responsive and interactive computing environment. They enhance resource utilization, enable efficient multitasking, and promote user collaboration and productivity.

21. The difference between Process and programme.-5

ANS

Program	Process
Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
Program is a static entity.	Process is a dynamic entity.
Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
Program does not have any control block.	Process has its own control block called Process Control Block.

Program has two logical components: code and data.	In addition to program data, a process also requires additional information required for the management and execution.
Program does not change itself.	Many processes may execute a single program. There program code may be the same but program data may be different. these are never same.
Program is a passive entity.	Process is active entity.
Program contains instructions	Process is a sequence of instruction execution.
A program exists at single place and continues to exist.	Process exists in a limited span of time.
Program is a static entity.	Process is a dynamic entity.

22. generations of operating system detail.-5

ANS Operating systems have evolved over time, and they are typically classified into different generations based on their characteristics and technological advancements. Here's an overview of the different generations of operating systems:

1. First Generation (1940s-1950s):

- The first generation of operating systems was primarily used in early computers like ENIAC and UNIVAC.
- These systems were extremely primitive and had no interactive interface.

- They relied on machine language programming and used punched cards or tape for input and output.

- Batch processing was the common mode of operation, where a set of jobs were collected and processed together.

2. Second Generation (1950s-1960s):

- The second generation of operating systems introduced the concept of multiprogramming, enabling the execution of multiple programs concurrently.

- It brought advancements like the introduction of assembly languages and the development of batch systems with the use of magnetic tapes for data storage.

- Operating systems of this generation focused on improving the efficiency of resource utilization and optimizing CPU utilization.

3. Third Generation (1960s-1970s):

- The third generation of operating systems saw significant advancements in terms of features and capabilities.

- Time-sharing systems were introduced, enabling multiple users to simultaneously access a computer system through terminals.

- Interactive interfaces and high-level programming languages like COBOL and FORTRAN were developed.

- This generation also witnessed the development of multiprogramming systems with advanced features such as virtual memory and file systems.

4. Fourth Generation (1970s-1980s):

- The fourth generation of operating systems witnessed the emergence of personal computers (PCs) and microprocessors.

- These systems featured graphical user interfaces (GUIs) and introduced the concept of multitasking, allowing users to run multiple applications simultaneously.

- Operating systems like MS-DOS, Mac OS, and early versions of UNIX were prevalent during this era.

5. Fifth Generation (1990s-Present):

- The fifth generation of operating systems focused on providing enhanced performance, scalability, and networking capabilities.

- Graphical operating systems, such as Windows 95, Windows NT, and macOS, became dominant.

- Client-server architecture and distributed computing gained popularity, enabling networked systems and internet connectivity.

- This generation also saw the rise of mobile operating systems like iOS and Android, designed specifically for smartphones and tablets.

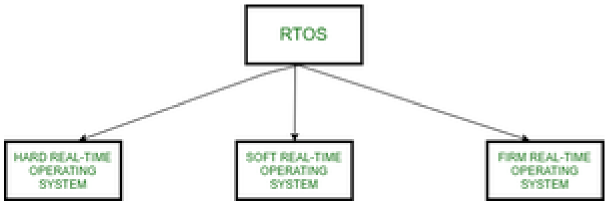
It's worth noting that the classification into generations is a broad categorization, and there is no strict boundary between them. Operating systems continue to evolve with ongoing advancements in technology, incorporating features like virtualization, cloud computing, real-time processing, and improved security measures.

23. Real-Time Operating System.-5

ANS Real-time **operating systems (RTOS)** are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines. such applications are industrial control, telephone switching equipment, flight control, and real-time simulations. With an RTOS, the processing time is measured in tenths of seconds.

This system is time-bound and has a fixed deadline. The processing in this type of system must occur within the specified constraints. Otherwise, This will lead to system failure.

Examples of the real-time operating systems: Airline traffic control systems, Command Control Systems, Airlines reservation system, Heart Pacemaker, Network Multimedia Systems, Robot etc.
The real-time operating systems can be of 3 types –


RTOS
HARD REAL-TIME OPERATING SYSTEM
SOFT REAL-TIME OPERATING SYSTEM
FIRM REAL-TIME OPERATING SYSTEM

12. **Hard Real-Time operating system:**
    These operating systems guarantee that critical tasks be completed within a range of time.
13. For example, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete car welding by robot hardly on the time., scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

14. **Soft real-time operating system:**
    This operating system provides some relaxation in the time limit.
15. For example – Multimedia systems, digital audio systems etc. Explicit, programmer-defined and controlled processes are encountered in real-time systems. A separate process is changed with handling a single external event. The process is activated upon occurrence of the related event signalled by an interrupt.
16. Multitasking operation is accomplished by scheduling processes for execution independently of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services. The processor is allocated to the highest priority processes. This type of schedule, called, priority-based preemptive scheduling is used

by real-time systems.

17. **Firm Real-time Operating System**:
RTOS of this type have to follow deadlines as well. In spite of its small impact, missing a deadline can have unintended consequences, including a reduction in the quality of the product. Example: Multimedia applications.
18.

Advantages:

The advantages of real-time operating systems are as follows-

19. **Maximum consumption –**
Maximum utilization of devices and systems. Thus more output from all the resources.

20. **Task Shifting –**
Time assigned for shifting tasks in these systems is very less. For example, in older systems, it takes about 10 microseconds. Shifting one task to another and in the latest systems, it takes 3 microseconds.

21. **Focus On Application –**
Focus on running applications and less importance to applications that are in the queue.

22. **Real-Time Operating System In Embedded System –**
Since the size of programs is small, RTOS can also be embedded systems like in transport and others.

23. **Error Free –**
These types of systems are error-free.

24. **Memory Allocation –**
Memory allocation is best managed in these types of systems.

Disadvantages:

The disadvantages of real-time operating systems are as follows-

25. **Limited Tasks –**
    Very few tasks run simultaneously, and their concentration is very less on few applications to avoid errors.

26. **Use Heavy System Resources –**
    Sometimes the system resources are not so good and they are expensive as well.

27. **Complex Algorithms –**
    The algorithms are very complex and difficult for the designer to write on.

28. **Device Driver And Interrupt signals –**
    It needs specific device drivers and interrupts signals to respond earliest to interrupts.

29. **Thread Priority –**
    It is not good to set thread priority as these systems are very less prone to switching tasks.

30. **Minimum Switching –** RTOS performs minimal task switching.

Comparison of Regular and Real-Time operating systems:

Regular OS	Real-Time OS (RTOS)
Complex	Simple
Best effort	Guaranteed response
Fairness	Strict Timing constraints
Average Bandwidth	Minimum and maximum limits
Unknown components	Components are known

Unpredictable behavior	Predictable behavior
Plug and play	RTOS is upgradeable

24. context switching with a diagram.-5

ANS An operating system is a program loaded into a system or computer. and manage all the other program which is running on that OS Program, it manages the all other application programs. or in other words, we can say that the OS is an interface between the user and computer hardware.

So in this article, we will learn about what is Context switching in an Operating System and see how it works also understands the triggers of context switching and an overview of the Operating System.

Context Switch
Context switching in an operating system involves saving the context or state of a running process so that it can be restored later, and then loading the context or state of another. process and run it.

Context Switching refers to the process/method used by the system to change the process from one state to another using the CPUs present in the system to perform its job.

Example – Suppose in the OS there (N) numbers of processes are stored in a Process Control Block(PCB). like The process is running using the CPU to do its job. While a process is running, other processes with the highest priority queue up to use the CPU to complete their job.

Process Control Block
So, The Process Control block(PCB) is also known as a Task Control Block. it represents a process in the Operating System. A process control block

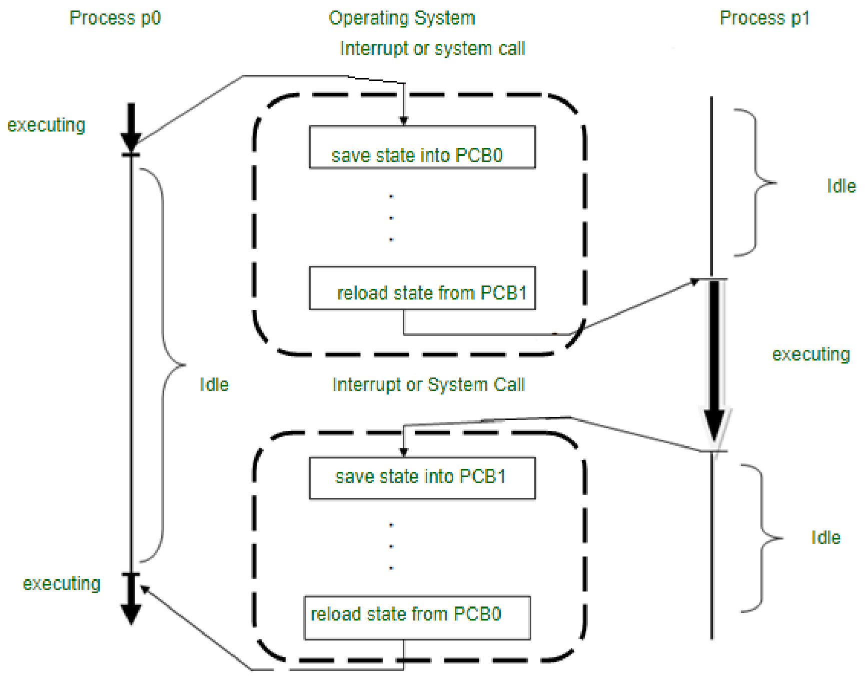
(PCB) is a data structure used by a computer to store all information about a process. It is also called the descriptive process. When a process is created (started or installed), the operating system creates a process manager.

Context Switching Cases

So basically context switching can be applicable to triggered by various events, and properties in OS. which contains some major triggers as-

31. **Interrupt** – To handle the interrupt OS performs a context switch to the handler to handle an interrupt and switch back to the origin process.
32. **Multitasking** – Context switching is the hallmark of multitasking. when various processes are running on the system then OS switches allow each process to execute for a particular time period, that time is known as a time slice.
33. **System Calls** – When a process makes any system calls, the OS switches the mode of the kernel and saves that process in context, and executes the system call.
34. **Kernel/User Switch** – This trigger is used when the OS needed to switch between the user mode and kernel mode.

State Diagram of Context Switching



Working Process Context Switching

So the context switching of two processes, the priority-based process occurs in the ready queue of the process control block. These are the following steps.

- The state of the current process must be saved for rescheduling.
- The process state contains records, credentials, and operating system-specific information stored on the PCB or switch.
- The PCB can be stored in a single layer in kernel memory or in a custom OS file.
- A handle has been added to the PCB to have the system ready to run.
- The operating system aborts the execution of the current process and selects a process from the waiting list by tuning its PCB.
- Load the PCB's program counter and continue execution in the selected process.

- Process/thread values can affect which processes are selected from the queue, this can be important.

25. Disk Arm Scheduling Algorithm.-5

ANS Disk arm scheduling algorithms determine the order in which disk requests are serviced by the disk arm. Here are commonly used algorithms:

1. FCFS (First-Come, First-Served): Requests are serviced in the order they arrive. However, this algorithm can result in poor performance if requests are scattered across the disk.

2. SSTF (Shortest Seek Time First): Selects the request that requires the shortest seek time from the current position. It aims to minimize arm movement and reduce access time.

3. SCAN (Elevator) Algorithm: The disk arm moves from one end of the disk to the other, servicing requests along the way. It provides fair servicing but requests farthest from the current position may wait longer.

4. C-SCAN (Circular SCAN): Similar to SCAN but jumps to the other end instead of reversing direction. Ensures requests are always serviced in one direction, avoiding long wait times.

5. LOOK Algorithm: Similar to SCAN, but stops at the last request in each direction. Reverses movement to reduce arm movement and improve performance.

These algorithms optimize disk access time by reducing seek time and maximizing throughput. The choice depends on the workload characteristics and performance requirements.