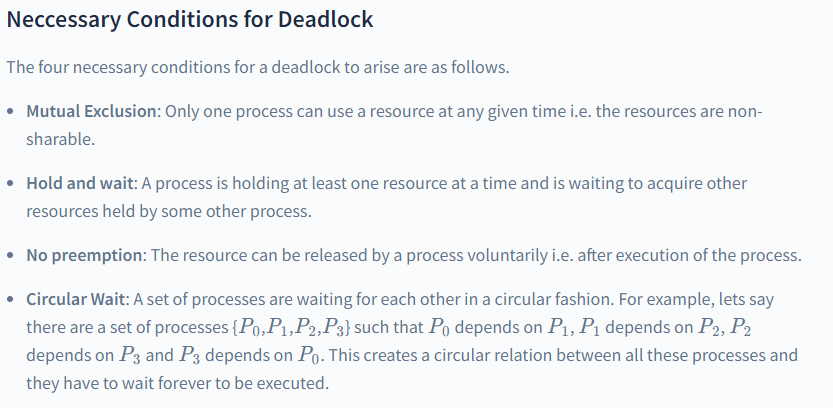
[**Notes --------🡪 click**](https://github.com/neeraj46665/BTech-CS-Notes/tree/main/BTech-2nd-year/sem-4)

1. Explain the deadlock. Discuss the necessary conditions for deadlock?

All the processes in a system require some resources such as central processing unit(CPU), file storage, input/output devices, etc to execute it. Once the execution is finished, the process releases the resource it was holding. However, when many processes run on a system they also compete for these resources they require for execution. This may arise a deadlock situation.

A **deadlock** is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process. Therefore, none of the processes gets executed.

****

prevention

### Deadlock Prevention

This is done by restraining the ways a request can be made. Since deadlock occurs when all the above four conditions are met, we try to prevent any one of them, thus preventing a deadlock.

### Deadlock Avoidance

When a process requests a resource, the deadlock avoidance algorithm examines the resource-allocation state. If allocating that resource sends the system into an unsafe state, the request is not granted.

Therefore, it requires additional information such as how many resources of each type is required by a process. If the system enters into an unsafe state, it has to take a step back to avoid deadlock.

### Deadlock Detection and Recovery

We let the system fall into a deadlock and if it happens, we detect it using a detection algorithm and try to recover.

**Some ways of recovery are as follows.**

* Aborting all the deadlocked processes.
* Abort one process at a time until the system recovers from the deadlock.
* Resource Preemption: Resources are taken one by one from a process and assigned to higher priority processes until the deadlock is resolved.

### Deadlock Ignorance

In the method, the system assumes that deadlock never occurs. Since the problem of deadlock situation is not frequent, some systems simply ignore it. Operating systems such as UNIX and Windows follow this approach. However, if a deadlock occurs we can reboot our system and the deadlock is resolved automatically.

**Note:** The above approach is an example of Ostrich Algorithm. It is a strategy of ignoring potential problems on the basis that they are extremely rare.

1. Explain the concept of multi-threading. Discuss the following multi-threading models.
   1. Many-to-one
   2. One-to-one
   3. Many-to-many

Multithreading allows the application to divide its task into individual threads. In multi-threads, the same process or task can be done by the number of threads, or we can say that there is more than one thread to perform the task in multithreading. With the use of multithreading, multitasking can be achieved.



The main drawback of single threading systems is that only one task can be performed at a time, so to overcome the drawback of this single threading, there is multithreading that allows multiple tasks to be performed.

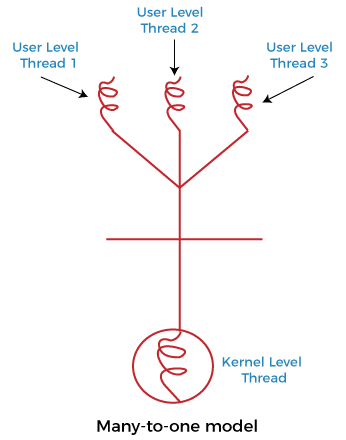
**There exists three established multithreading models classifying these relationships are:**

* Many to one multithreading model
* One to one multithreading model
* Many to Many multithreading models

### **Many to one multithreading model:**

The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment, easily implemented even on the simple kernel with no thread support.

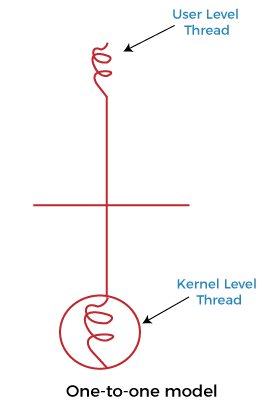
The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware acceleration offered by multithreaded processes or multi-processor systems. In this, all the thread management is done in the userspace. If blocking comes, this model blocks the whole system.



In the above figure, the many to one model associates all user-level threads to single kernel-level threads.

### **One to one multithreading model**

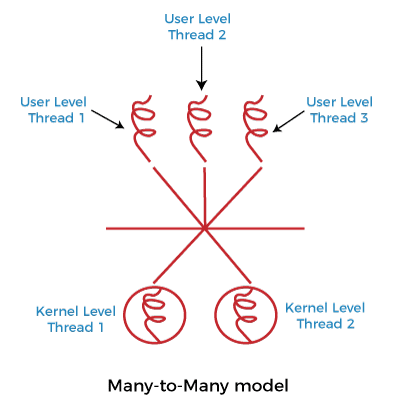
The one-to-one model maps a single user-level thread to a single kernel-level thread. This type of relationship facilitates the running of multiple threads in parallel. However, this benefit comes with its drawback. The generation of every new user thread must include creating a corresponding kernel thread causing an overhead, which can hinder the performance of the parent process. Windows series and Linux operating systems try to tackle this problem by limiting the growth of the thread count.



In the above figure, one model associates that one user-level thread to a single kernel-level thread.

### **Many to Many Model multithreading model**

In this type of model, there are several user-level threads and several kernel-level threads. The number of kernel threads created depends upon a particular application. The developer can create as many threads at both levels but may not be the same. The many to many model is a compromise between the other two models. In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution. Also, with the introduction of multiple threads, complexity is not present as in the previous models. Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model. This is because the kernel can schedule only one process at a time.



1. Describe in brief - the concept of fragmentation.

# **Fragmentation in Operating System**

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused.

Contiguous memory allocation allocates space to processes whenever the processes enter **RAM**. These **RAM** spaces are divided either by fixed partitioning or by dynamic partitioning. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to coming processes.

## What is Fragmentation?

Fragmentation is an unwanted problem in the [operating system](https://www.javatpoint.com/os-tutorial) in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called **fragmentation**.

## Causes of Fragmentation

User processes are loaded and unloaded from the main memory, and processes are kept in memory blocks in the main memory. Many spaces remain after process loading and swapping that another process cannot load due to their size. Main memory is available, but its space is insufficient to load another process because of the dynamical allocation of main memory processes.

## Types of Fragmentation

There are mainly two types of fragmentation in the operating system. These are as follows:

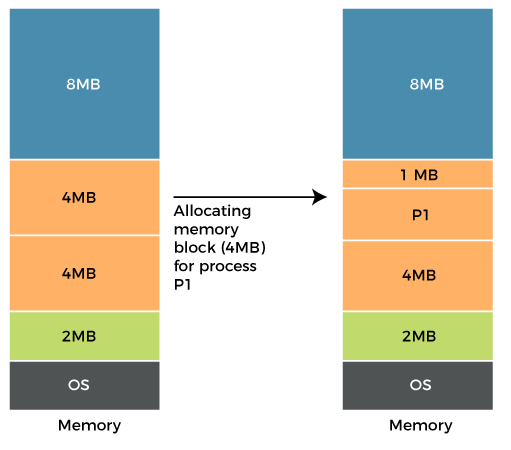
1. **Internal Fragmentation**
2. **External Fragmentation**

### **Internal Fragmentation**

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes **internal** fragmentation.

**For Example:**

Assume that memory allocation in RAM is done using fixed partitioning (i.e., memory blocks of fixed sizes). **2MB, 4MB, 4MB**, and **8MB** are the available sizes. The Operating System uses a part of this RAM.



Let's suppose a process **P1** with a size of **3MB** arrives and is given a memory block of **4MB**. As a result, the **1MB** of free space in this block is unused and cannot be used to allocate memory to another process. It is known as **internal fragmentation**.

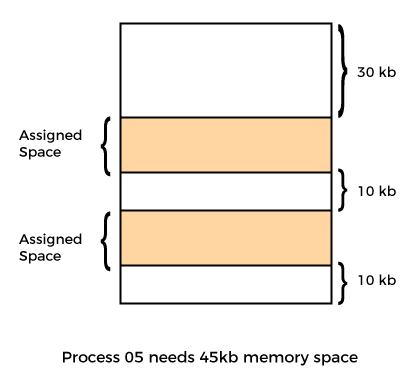
**How to avoid internal fragmentation?**

The problem of internal fragmentation may arise due to the fixed sizes of the memory blocks. It may be solved by assigning space to the process via dynamic partitioning. Dynamic partitioning allocates only the amount of space requested by the process. As a result, there is no internal fragmentation.

### **External Fragmentation**

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as **external** fragmentation.

**For Example:**



Let's take the example of external fragmentation. In the above diagram, you can see that there is sufficient space **(50 KB)** to run a process **(05) (need 45KB)**, but the memory is not contiguous. You can use compaction, paging, and segmentation to use the free space to execute a process.

**How to remove external fragmentation?**

This problem occurs when you allocate RAM to processes continuously. It is done in paging and segmentation, where memory is allocated to processes non-contiguously. As a result, if you remove this condition, external fragmentation may be decreased.

Compaction is another method for removing external fragmentation. External fragmentation may be decreased when dynamic partitioning is used for memory allocation by combining all free memory into a single large block. The larger memory block is used to allocate space based on the requirements of the new processes. This method is also known as defragmentation.

## Advantages and disadvantages of fragmentation

There are various advantages and disadvantages of fragmentation. Some of them are as follows:

### **Advantages**

There are various advantages of fragmentation. Some of them are as follows:

**Fast Data Writes**

Data write in a system that supports data fragmentation may be faster than reorganizing data storage to enable contiguous data writes.

**Fewer Failures**

If there is insufficient sequential space in a system that does not support fragmentation, the write will fail.

**Storage Optimization**

A fragmented system might potentially make better use of a storage device by utilizing every available storage block.

### **Disadvantages**

There are various disadvantages of fragmentation. Some of them are as follows:

**Need for regular defragmentation**

A more fragmented storage device's performance will degrade with time, necessitating the requirement for time-consuming defragmentation operations.

**Slower Read Times**

The time it takes to read a non-sequential file might increase as a storage device becomes more fragmented.

## Conclusion

In short, both internal and external fragmentation are natural processes that result in either memory wasting or empty memory space. However, the problems in both cases cannot be completely overcome, although they can be reduced to some extent using the solutions provided above.

1. Define the term- Bare Machine.

Bare machine is logical hardware which is used to execute the program in the processor without using the operating system. as of now, we have studied that we can’t execute any process without the Operating system. But yes with the help of the Bare machine we can do that. Initially, when the operating systems are not developed, the execution of an instruction is done by directly on hardware without using any interfering hardware, at that time the only drawback was that the Bare machines accepting the instruction in only machine language, due to this those person who has sufficient knowledge about Computer field are able to operate a computer. so after the development of the operating system Bare machine is referred to as inefficient.

1. Briefly discuss holes in memory partitioning.
2. Explain control language interpreter and Loader in reference to Resident monitor.

Resident Monitor:

if we talk about how the code runs on Bare machines, then this component is used, so basically, the Resident Monitor is a code that runs on Bare Machines.

The resident monitor works like an operating system that controls the instructions and performs all necessary functions. It also works like job sequencer because it also sequences the job and sends them to the processor.

After scheduling the job Resident monitors loads the programs one by one into the main memory according to their sequences. One most important factor about the resident monitor is that when the program execution occurred there is no gap between the program execution and the processing is going to be faster.

The Resident monitors are divided into 4 parts as:

1. Control Language Interpreter

2. Loader

These are explained as following below.

Control Language Interpreter:

The first part of the Resident monitor is control language interpreter which is used to read and carry out the instruction from one level to the next level.

Loader:

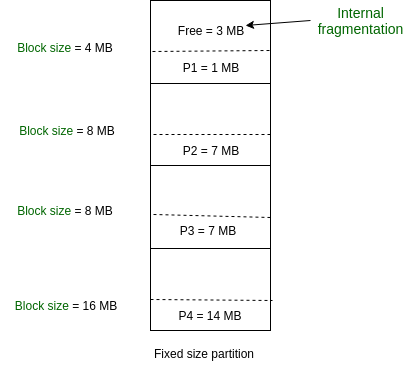
The second part of the Resident monitor which is the main part of the Resident Monitor is Loader which Loads all the necessary system and application programs into the main memory.

1. Discuss about static partitioning.

Fixed partitioning, also known as static partitioning, is a memory allocation technique used in operating systems to divide the physical memory into fixed-size partitions or regions, each assigned to a specific process or user. Each partition is typically allocated at system boot time and remains dedicated to a specific process until it terminates or releases the partition.

Fixed Partitioning:

This is the oldest and simplest technique used to put more than one process in the main memory. In this partitioning, the number of partitions (non-overlapping) in RAM is fixed but the size of each partition may or may not be the same. As it is a contiguous allocation, hence no spanning is allowed. Here partitions are made before execution or during system configure.



As illustrated in above figure, first process is only consuming 1MB out of 4MB in the main memory.

Hence, Internal Fragmentation in first block is (4-1) = 3MB.

Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

Suppose process P5 of size 7MB comes. But this process cannot be accommodated in spite of available free space because of contiguous allocation (as spanning is not allowed). Hence, 7MB becomes part of External Fragmentation.

**Advantages of Fixed Partitioning –**

Easy to implement:

Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into a certain partition without focusing on the emergence of Internal and External Fragmentation.

Little OS overhead:

Processing of Fixed Partitioning requires lesser excess and indirect computational power.

Predictable and can ensure a minimum amount of memory for each process.

Can prevent processes from interfering with each other’s memory space.

Suitable for systems with a fixed number of processes and known memory requirements.

**Disadvantages of Fixed Partitioning –**

Internal Fragmentation:

Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.

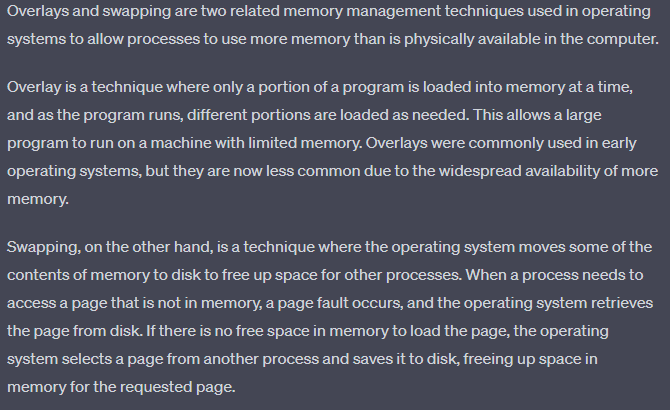
External Fragmentation:

The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).

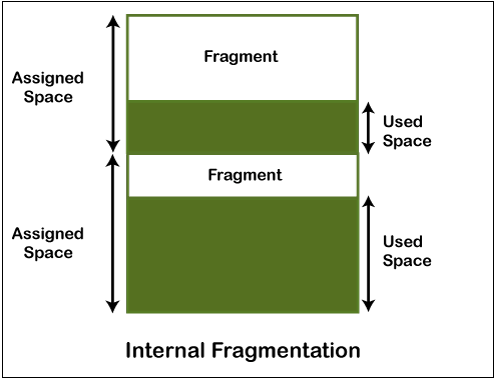
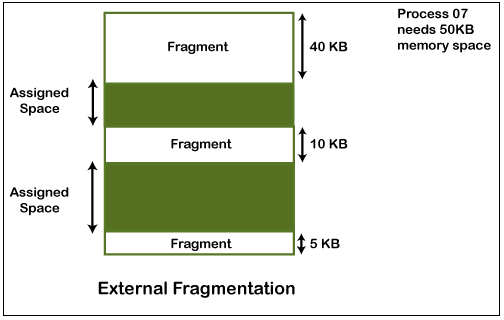
Limit process size:

Process of size greater than the size of the partition in Main Memory cannot be accommodated. The partition size cannot be varied according to the size of the incoming process size. Hence, the process size of 32MB in the above-stated example is invalid.

1. Write the short note on
   1. i.Overlays
   2. ii.Swapping

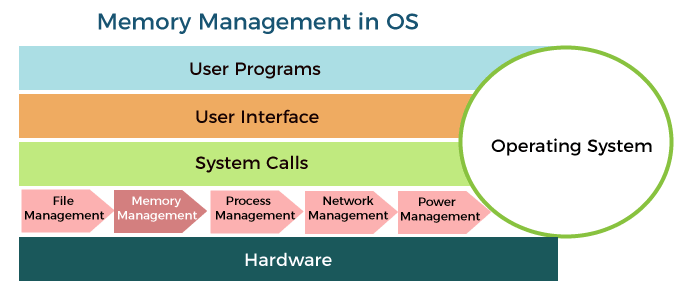


1. Illustrate with the help of diagram: internal and external fragmentation.

1. Explain memory management and its requirements.

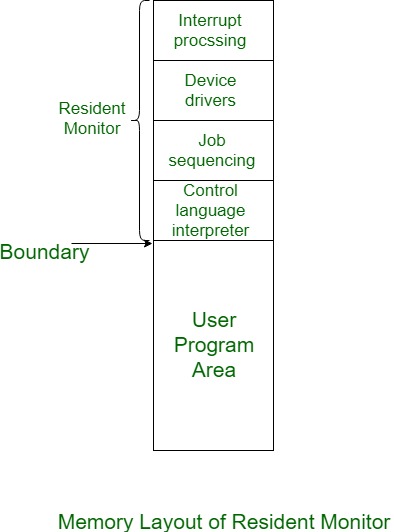
Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.

1. 

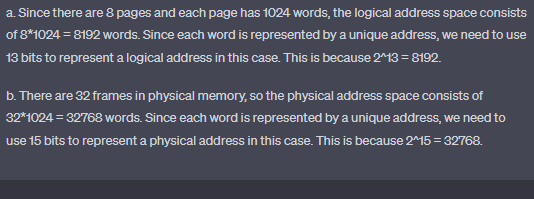
**Memory management requirements:**  
  
**1. Relocation**  
**2. Protection**  
**3. Sharing**  
**4. Logical Organization**  
**5. Physical Organization**  
  
  
  
**1. Relocation:**  
The programmer does not know where the program will be placed in memory when it is executed while the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated). Memory references must be translated into the code to the actual physical memory addresses.  
  
The address generated by the CPU is said to be a logical address. An address generated by MMU is called a physical address.   
  
  
**2. Protection:**  
Processes should not be able to reference memory locations in another process without permission. It must be checked at run time. The memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software).  
  
The word protection means provide security from unauthorized usage of memory. The operating can protect the memory with the help of base and limit register. Base registers consisting of the starting address of the next process. The limit specifies the boundary of that job, so the limit register is also said to be a fencing register.  
  
  
**3. Sharing:**  
Allow several processes to access the same portion of memory. It is better to allow each process access to the same copy of the program rather than have their own separate copy.  
  
  
**4. Logical organization:**  
Programs are written in modules. Modules can be written and compiled independently. Different degrees of protection given to modules (read-only, execute-only). Modules are shared among processes.  
  
  
**5. Physical organization:**  
Memory available for a program plus its data may be insufficient. Overlaying allows various modules to be assigned to the same region of memory.

1. Illustrate the purpose of Device Driver in Resident monitors.

**Device Driver:**   
The third part of the Resident monitor is Device Driver which is used to manage the connecting input-output devices to the system. So basically it is the interface between the user and the system. it works as an interface between the request and response. request which user made, Device driver responds that the system produces to fulfill these requests.



1. Explain memory compaction and ways to implement swapping in detail.
2. Consider a logical address space of eight pages of 1024 words, each mapped onto a physical memory of 32 frames then :
   1. How many bits are in logical address?
   2. How many bits are in physical address?



1. Calculate the total head movement for the following scheduling.
   * + 1. FCFS
       2. SSTF
       3. SCAN
2. What do you understand by deadlock? ans1
3. What are the necessary conditions for deadlock? ans1
4. How can pre-emption be used to resolve deadlock?

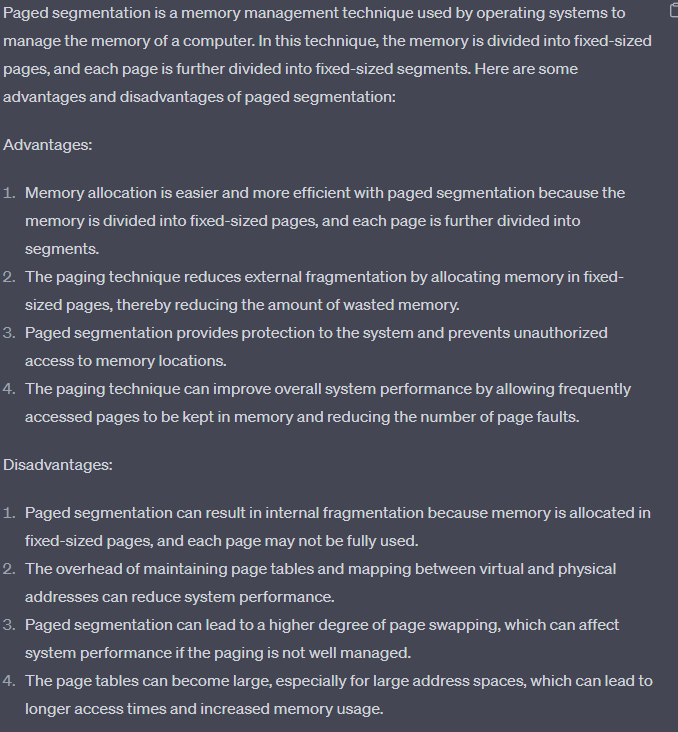
**2. Resource Preemption:**   
To eliminate deadlocks using resource preemption, we preempt some resources from processes and give those resources to other processes. This method will raise three issues –

* **(a). Selecting a victim:**   
  We must determine which resources and which processes are to be preempted and also the order to minimize the cost.
* **(b). Rollback:**   
  We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means abort the process and restart it.
* **(c). Starvation:**   
  In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

1. Write a difference between paging and segmentation?

| **S.NO** | **Paging** | **Segmentation** |
| --- | --- | --- |
| 1. | In paging, the program is divided into fixed or mounted size pages. | In segmentation, the program is divided into variable size sections. |
| 2. | For the paging operating system is accountable. | For segmentation compiler is accountable. |
| 3. | Page size is determined by hardware. | Here, the section size is given by the user. |
| 4. | It is faster in comparison to segmentation. | Segmentation is slow. |
| 5. | Paging could result in internal fragmentation. | Segmentation could result in external fragmentation. |
| 6. | In paging, the logical address is split into a page number and page offset. | Here, the logical address is split into section number and section offset. |
| 7. | Paging comprises a page table that encloses the base address of every page. | While segmentation also comprises the segment table which encloses segment number and segment offset. |
| 8. | The page table is employed to keep up the page data. | Section Table maintains the section data. |
| 9. | In paging, the operating system must maintain a free frame list. | In segmentation, the operating system maintains a list of holes in the main memory. |
| 10. | Paging is invisible to the user. | Segmentation is visible to the user. |
| 11. | In paging, the processor needs the page number, and offset to calculate the absolute address. | In segmentation, the processor uses segment number, and offset to calculate the full address. |
| 12. | It is hard to allow sharing of procedures between processes. | Facilitates sharing of procedures between the processes. |
| 13 | In paging, a programmer cannot efficiently handle data structure. | It can efficiently handle data structures. |
| 14. | This protection is hard to apply. | Easy to apply for protection in segmentation. |
| 15. | The size of the page needs always be equal to the size of frames. | There is no constraint on the size of segments. |
| 16. | A page is referred to as a physical unit of information. | A segment is referred to as a logical unit of information. |
| 17. | Paging results in a less efficient system. | Segmentation results in a more efficient system. |

1. List out the advantage and disadvantage of paged segmentation.



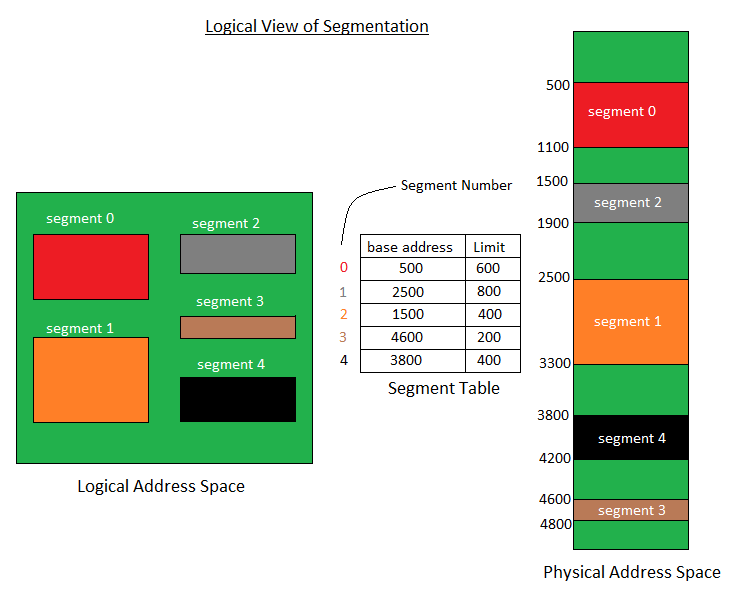
1. Illustrate Segmentation and define its types.

Segmentation divides processes into smaller subparts known as **modules**. The divided segments need not be placed in contiguous memory. Since there is no contiguous memory allocation, internal fragmentation does not take place. The length of the segments of the program and memory is decided by the purpose of the segment in the user program.

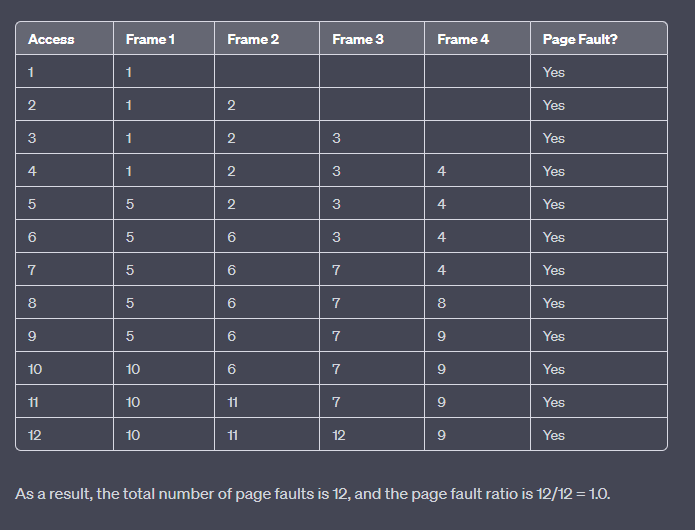
**Types of Segmentation in OS**

Segmentation in OS can be divided into two types:

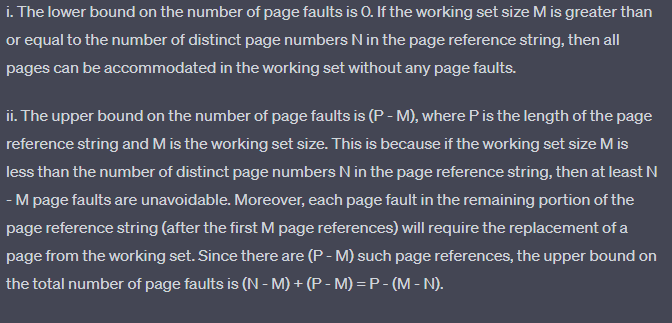
1. **Virtual Memory Segmentation**: Virtual Memory Segmentation divides the processes into **n** number of segments. All the segments are not divided at a time. Virtual Memory Segmentation may or may not take place at the run time of a program.
2. **Simple Segmentation**: Simple Segmentation also divides the processes into **n** number of segments but the segmentation is done all together at once. Simple segmentation takes place at the run time of a program. Simple segmentation may scatter the segments into the memory such that one segment of the process can be at a different location than the other(in a noncontinuous manner).



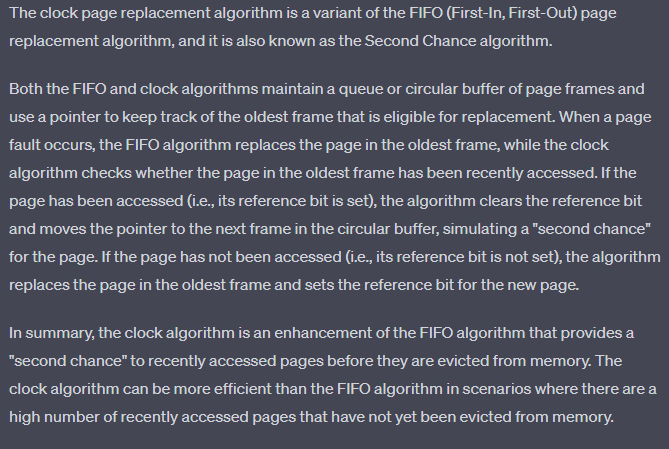
1. If the average page faults service time of 25 ms and a memory access time of 100ns.Calculate the effective access time.
2. Consider the following page-reference string: 1,2,3,4,5,6,7,8,9,10,11,12. How many page faults and page fault ratio would occur for the FIFO page replacement algorithm? Assuming there is four frames



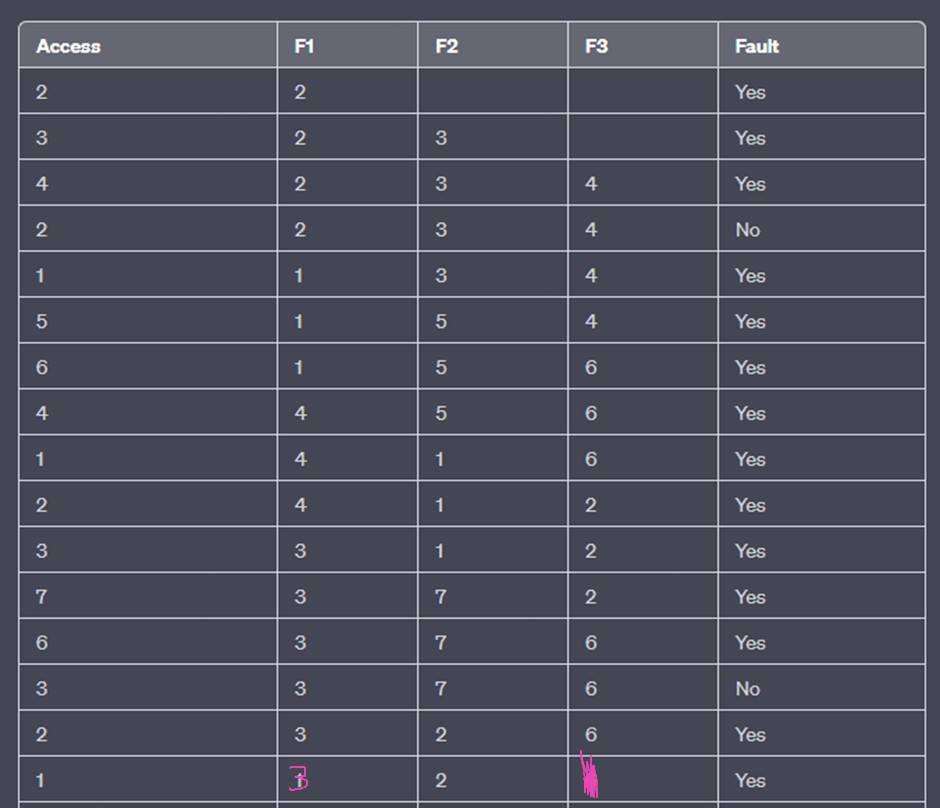
1. Give the steps required to handle a page fault in demand paging with example.ans31
2. Consider a page reference string for a process with a working set of M frames, initially all empty. The page reference string is of length P with N distinct page numbers in it. For any page replacement algorithm,
   * 1. What is a lower bound on the number of page faults?
     2. What is an upper bound on the number of page faults?



1. What is the relationship between FIFO and clock page replacement algorithm?



1. Consider the following page reference string: 2, 3, 4, 2, 1, 5, 6, 4, 1, 2, 3, 7, 6, 3, 2, 1 Calculate the number of page faults would occur for the LRU page replacement algorithm with frame size of 3.



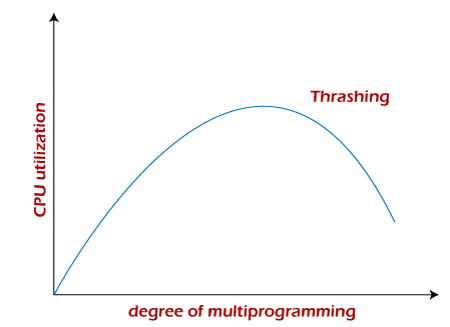
1. Evaluate when page faults will occur? Describe the actions taken by operating system during page fault.

 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 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.

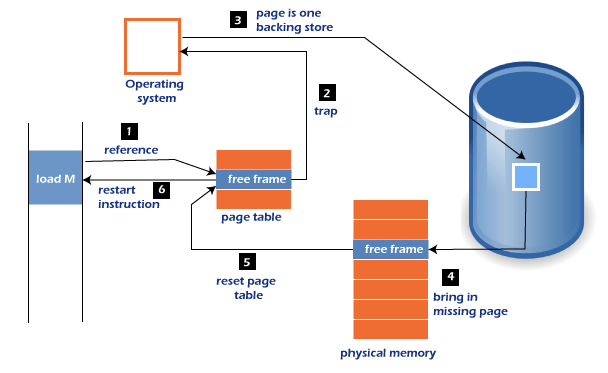
1. Define the Page Fault.

A page fault is an interruption that occurs when a software program attempts to access a memory block not currently stored in the system's RAM. This exception tells the operating system to find the block in virtual memory so it can be sent from a device's storage (SSD or HD) to RAM.

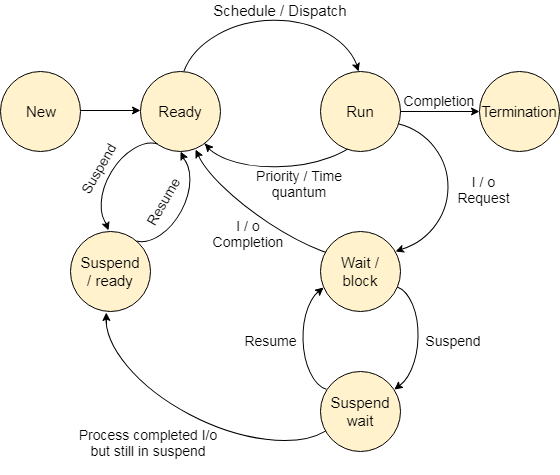
1. Discuss the Thrashing?

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. 

1. Discuss the steps needed to handle page fault with neat illustration.

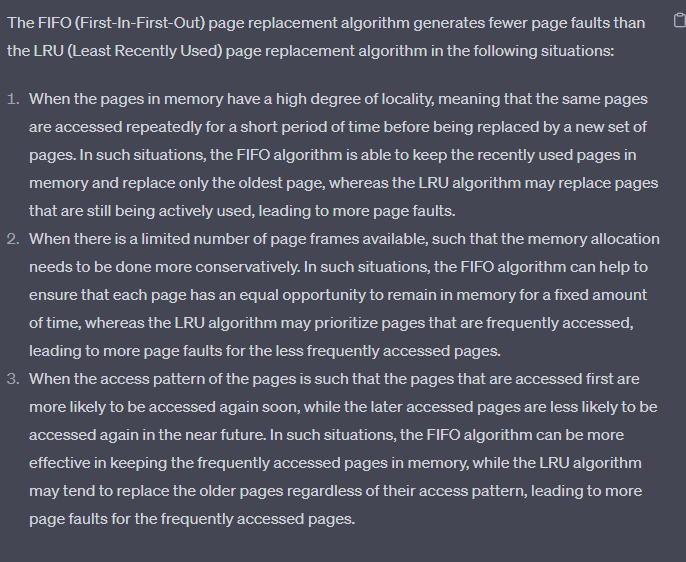


1. Firstly, an internal table for this process to assess whether the reference was valid or invalid memory access.
2. If the reference becomes invalid, the system process would be terminated. Otherwise, the page will be paged in.
3. After that, the free-frame list finds the free frame in the system.
4. Now, the disk operation would be scheduled to get the required page from the disk.
5. When the I/O operation is completed, the process's page table will be updated with a new frame number, and the invalid bit will be changed. Now, it is a valid page reference.
6. If any page fault is found, restart these steps from starting.
7. What is a process? Discuss components of process and various states of a process with the help of a process state transition diagram.



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

1. Discuss situation under which the FIFO page replacement algorithm generates fewer page faults than the LRU page replacement algorithm



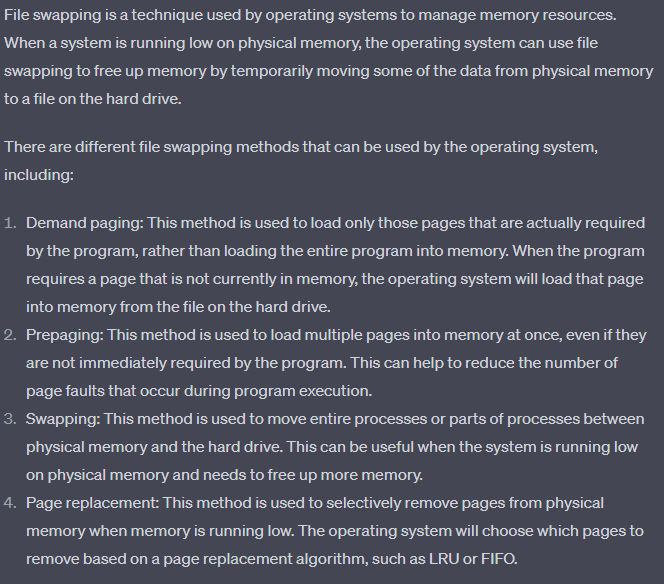
1. Illustrate what are the various Page Replacement Algorithms used in memory management.

Page Replacement Algorithms

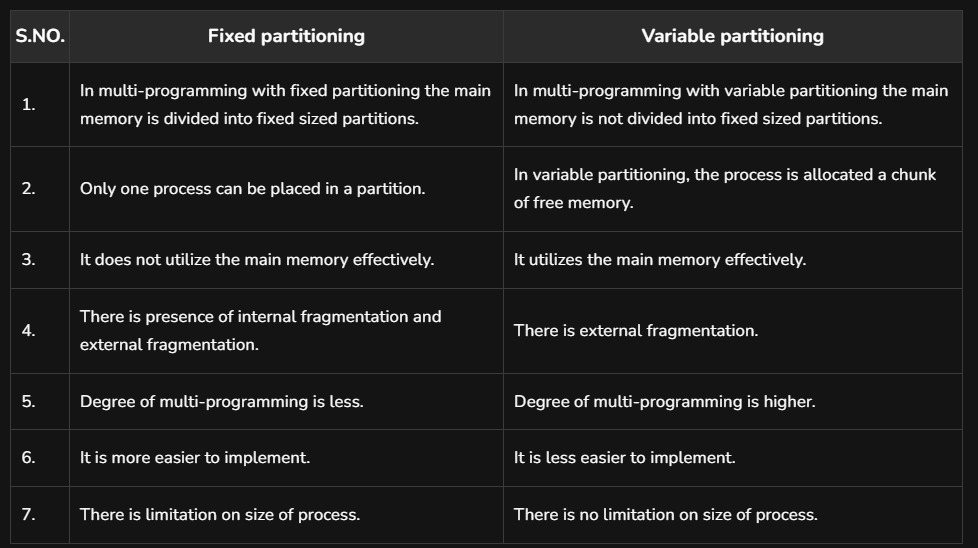
There are three types of Page Replacement Algorithms. They are:

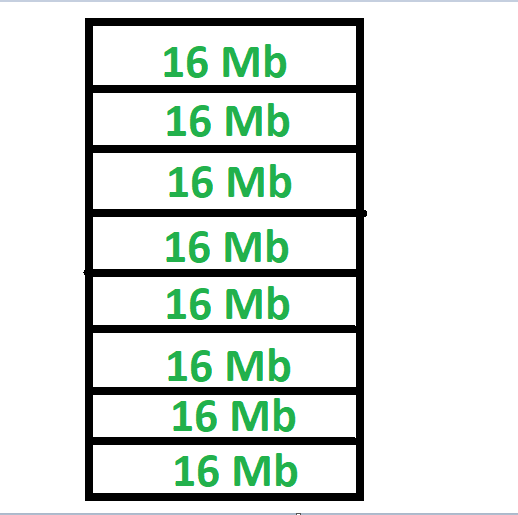
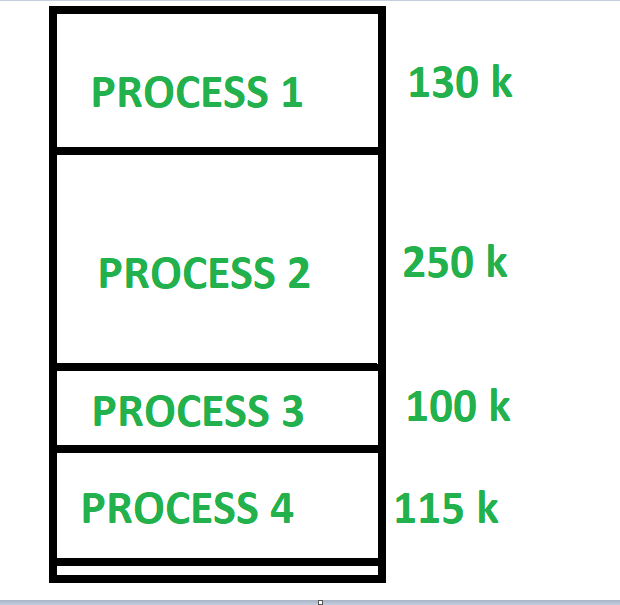
* Optimal Page Replacement Algorithm
* First In First Out Page Replacement Algorithm
* Least Recently Used (LRU) Page Replacement Algorithm

1. Consider the following page reference string: 1, 2, 3, 4, 5, 3,4,1,6,7,8,7, 8, 9, 7, 8, 9, 5, 4, 4, 5, 3 How many page faults would occur for the following replacement algorithms, assuming four frames? Remembering all frames are initially empty.
   * 1. FIFO replacement ii) Optimal replacement
2. Consider the following page reference string: 1,2, 3, 2, 5, 6, 3, 4, 6, 3, 7, 3, 1, 5, 3, 6, 3, 4, 2, 4, 3, 4, 5, 1 Indicate page faults and calculate total number of page faults and successful ratio for FIFO, optimal and LRU algorithms. Assume there are four frames and initially all the frames are empty.
3. What are the differences between paging and segmentation?ans19
4. Why are segmentation and paging sometimes combined into one Scheme?
5. Describe File swapping? List out the file swapping methods.



1. Under what condition(s) a wait state becomes a deadlock?
2. Analyze the concept of swapping and its implementation using different portioning methods.
3. Compare the two: multiprogramming with fixed partitioning and variable partitioning

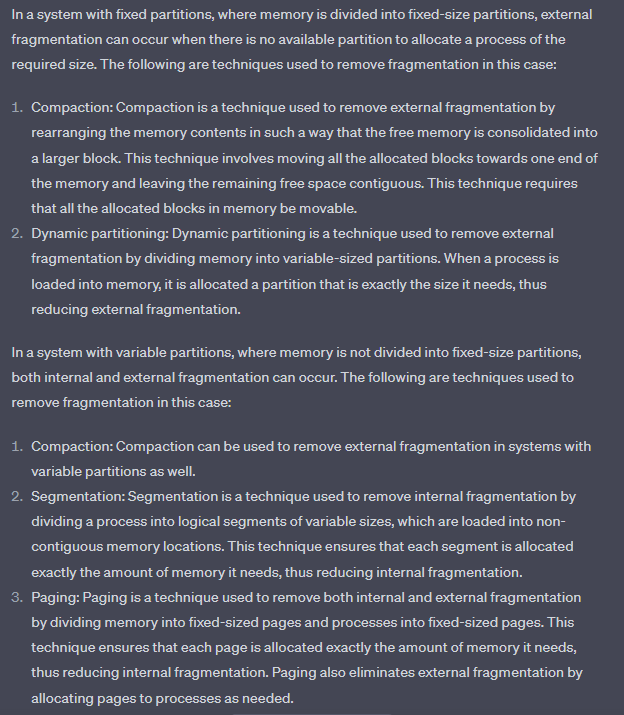


1. Compare all parts of resident monitors.

These are explained as following below.

1. **Control Language Interpreter:**   
   The first part of the Resident monitor is control language interpreter which is used to read and carry out the instruction from one level to the next level.
2. **Loader:**   
   The second part of the Resident monitor which is the main part of the Resident Monitor is Loader which Loads all the necessary system and application programs into the main memory.
3. **Device Driver:**   
   The third part of the Resident monitor is Device Driver which is used to manage the connecting input-output devices to the system. So basically it is the interface between the user and the system. it works as an interface between the request and response. request which user made, Device driver responds that the system produces to fulfill these requests.
4. **Interrupt Processing:**   
   The fourth part as the name suggests, it processes the all occurred interrupt to the system.
5. Justify the different techniques to remove fragmentation in case of multiprogramming with fixed partitions and variable partitions.



1. Given memory partition of 100k, 500k, 200k, 300k and 600k (in order). How would each of the first fit, best fit and worst fit algorithms place processes of 212 K, 417 K, 112 K and 426 K (in order)? Which algorithm makes the most efficient use of memory?

Answer:

1. First-fit
2. 212K is put in 500K partition
3. 417K is put in 600K partition
4. 112K is put in 288K partition(new partition 288K=500K-212K)
5. 426K must wait
6. Best-fit
7. 212K is put in 300K partition
8. 417K is put in 500K partition
9. 112K is put in 200K partition
10. 426K is put in 600K partition
11. Worst-fit
12. 212K is put in 600K partition
13. 417K is put in 500K partition
14. 112K is put in 388K partition
15. 426K must wait

In this example, Best-fit turns out to be the best.

1. Consider the following page-reference string:

1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

Calculate how many page faults would occur for the following

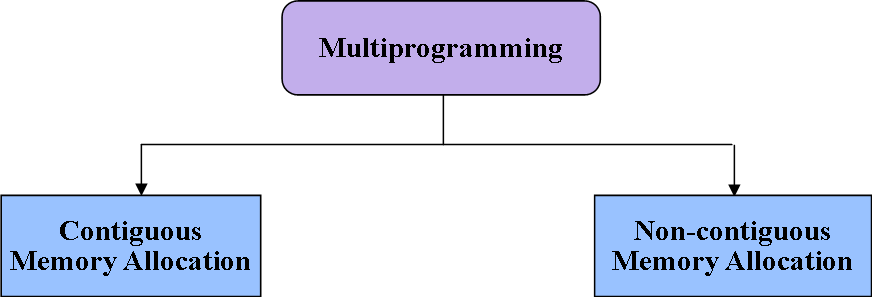
replacement algorithms, assuming the frame size to be 4. Assume that the

frames are initially empty.

* 1. LRU replacement
  2. FIFO replacement

1. Describe the memory management concept with advantages and disadvantages.

memory management is the process of controlling and coordinating a computer's main [memory](https://www.techtarget.com/whatis/definition/memory). It ensures that blocks of memory space are properly managed and allocated so the operating system ([OS](https://www.techtarget.com/whatis/definition/operating-system-OS)), [applications](https://www.techtarget.com/searchsoftwarequality/definition/application) and other running [processes](https://www.techtarget.com/whatis/definition/process) have the memory they need to carry out their operations.



#### **Advantages of segmented memory management technique**

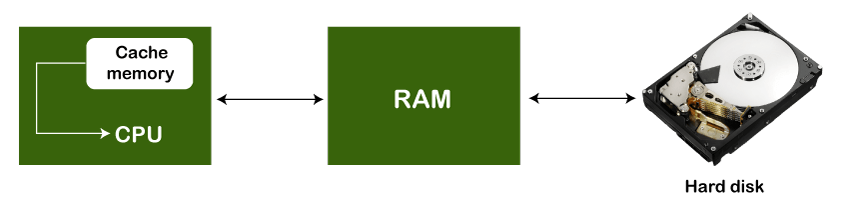
* Allow the memory ability to be 1 MB despite the fact that the addresses related with the individual directions are 16 bits wide.
* Allow the utilization of independent memory regions for the program code and information and stack part of the program.
* It permits a program and additionally its information to be put into various areas of memory at whatever point the program is end.
* Multitasking turns out to be simple

#### **Disadvantages of segmented memory management technique:**

* Availability of external fragmentation
* Algorithms of memory management are costly.
* Segmentation discovers free memory areas sufficiently large.
* Paging keeps rundown of free pages.
* Segments of inconsistent size not fit also for trading.

1. Illustrate the following?
   1. Virtual Memory
   2. Cache Memory
   3. Auxiliary Memory

|  |  |  |  |
| --- | --- | --- | --- |
| **S. N.** | **Parameter Difference** | **Cache Memory** | **Virtual Memory** |
| 1. | **Definition** | Cache Memory is the high speed of computer memory that reduces the access time of files or documents from the main memory. | Virtual Memory is a logical unit of computer memory that increases the capacity of main memory by storing or executing programs of larger size than the main memory in the computer system. |
| 2. | **Memory Unit** | Cache Memory is defined as a memory unit in a computer system. | Virtual Memory is not defined as a memory unit. |
| 3. | **Size** | Its size is very small as compared to Virtual Memory. | Its size is very large as compared to the Cache Memory. |
| 4. | **Speed** | It is a high-speed memory as compared to Virtual Memory. | It is not a high-speed memory as compared to the Cache Memory. |
| 5. | **Operation** | Generally, it stores frequently used data in the cache memory to reduce the access time of files. | The virtual memory keeps those data or programs that may not completely be placed in the main memory. |
| 6. | **Management** | Cache Memory is controlled by the hardware of a system. | Whereas the virtual memory is control by the Operating System (OS). |
| 7. | **Mapping** | It does not require a mapping structure to access the files in Cache Memory. | It requires a mapping structure to map the virtual address with a physical address. |

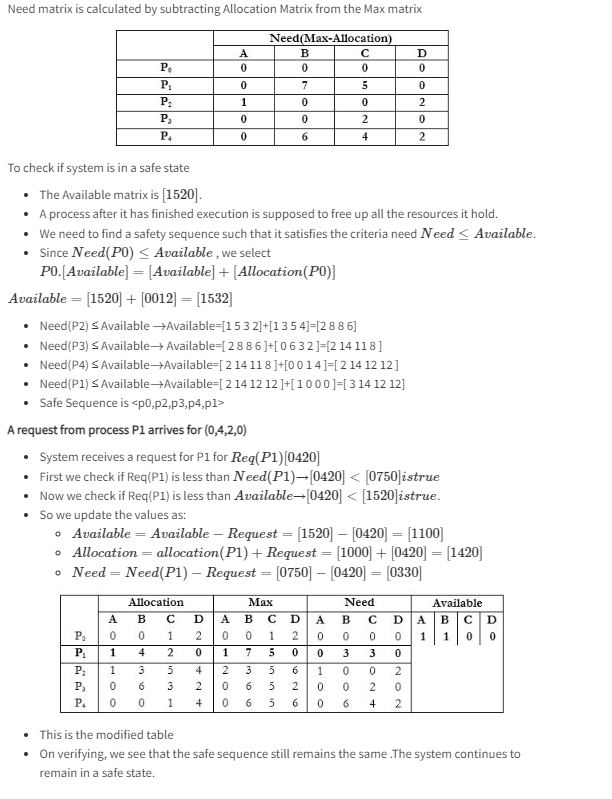


An Auxiliary memory is known as the lowest-cost, highest-capacity and slowest-access storage in a computer system. It is where programs and data are kept for long-term storage or when not in immediate use. The most common examples of auxiliary memories are magnetic tapes and magnetic disks.

1. Considering a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:

Analyze what will be the content of the Need matrix. Is the system in a safe state? If yes, then what is the safe sequence?

1. Illustrate the following.
   * 1. Memory Allocation Strategies
     2. Page table structure
     3. Deadlock Recovery
2. Show the page replacement policy with a suitable diagram. Why is it essential for the performance of an operating system?
3. Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in the main memory will
   1. Always decrease the number of page faults
   2. Always increase the number of page faults
   3. Sometimes increase the number of page faults
   4. Never affect the number of page faults
4. Justify your answer with an example.
5. A system uses 3-page frames for storing process pages in the main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults will occur while processing the page reference string given as 4, 7, 6, 1, 7, 6, 1, 2, 7, 2?
6. Answer the following questions using the banker’s algorithm:
   * + 1. What is the content of the matrix “Need”?
       2. Is the system in a safe state?
       3. If a request from process P1 arrives for (0, 4, 2, 0) can the request be granted immediately?



1. Discuss the Process Synchronization in Operating System.
2. Explain the paging in operating systems. Discuss with a diagram.
3. Suppose the moving head disk with 200 tracks is currently serving a request for track 85 and moving towards the larger cylinder number. If the sequence of all request come in the given order 98, 150, 101, 167, 120, 170.

# **Banker's Algorithm in Operating System (OS)**

It is a banker algorithm used to **avoid deadlock** and **allocate resources** safely to each process in the computer system. The '**S-State'** examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. In this section, we will learn the **Banker's Algorithm** in detail. Also, we will solve problems based on the **Banker's Algorithm**. To understand the Banker's Algorithm first we will see a real word example of it.

Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

Similarly, it works in an [**operating system**](https://www.javatpoint.com/operating-system). When a new process is created in a computer system, the process must provide all types of information to the [operating system](https://www.javatpoint.com/os-tutorial) like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as **deadlock avoidance algorithm** or **deadlock detection** in the operating system.

## Advantages

Following are the essential characteristics of the Banker's algorithm:

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

## Disadvantages

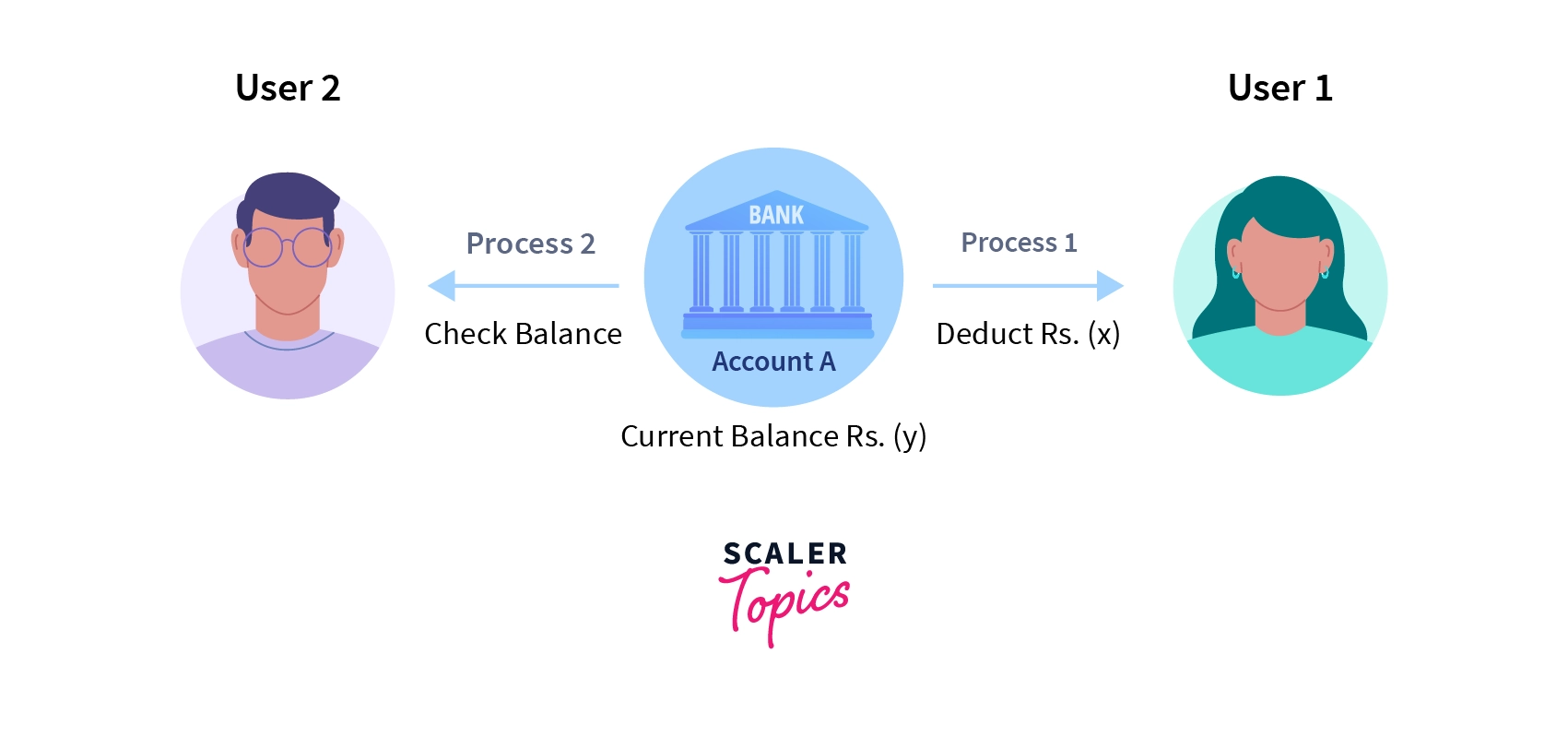
1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

Processes Synchronization or Synchronization is the way by which processes that share the same memory space are managed in an operating system. It helps maintain the consistency of data by using variables or hardware so that only one process can make changes to the shared memory at a time. There are various solutions for the same such as semaphores, mutex locks, synchronization hardware, etc.

**What is Process Synchronization in OS?**

An operating system is a software that manages all applications on a device and basically helps in the smooth functioning of our computer. Because of this reason, the operating system has to perform many tasks, and sometimes simultaneously. This isn't usually a problem unless these simultaneously occurring processes use a common resource.

For example, consider a bank that stores the account balance of each customer in the same database. Now suppose you initially have x rupees in your account. Now, you take out some amount of money from your bank account, and at the same time, someone tries to look at the amount of money stored in your account. As you are taking out some money from your account, after the transaction, the total balance left will be lower than x. But, the transaction takes time, and hence the person reads x as your account balance which leads to inconsistent data. If in some way, we could make sure that only one process occurs at a time, we could ensure consistent data.

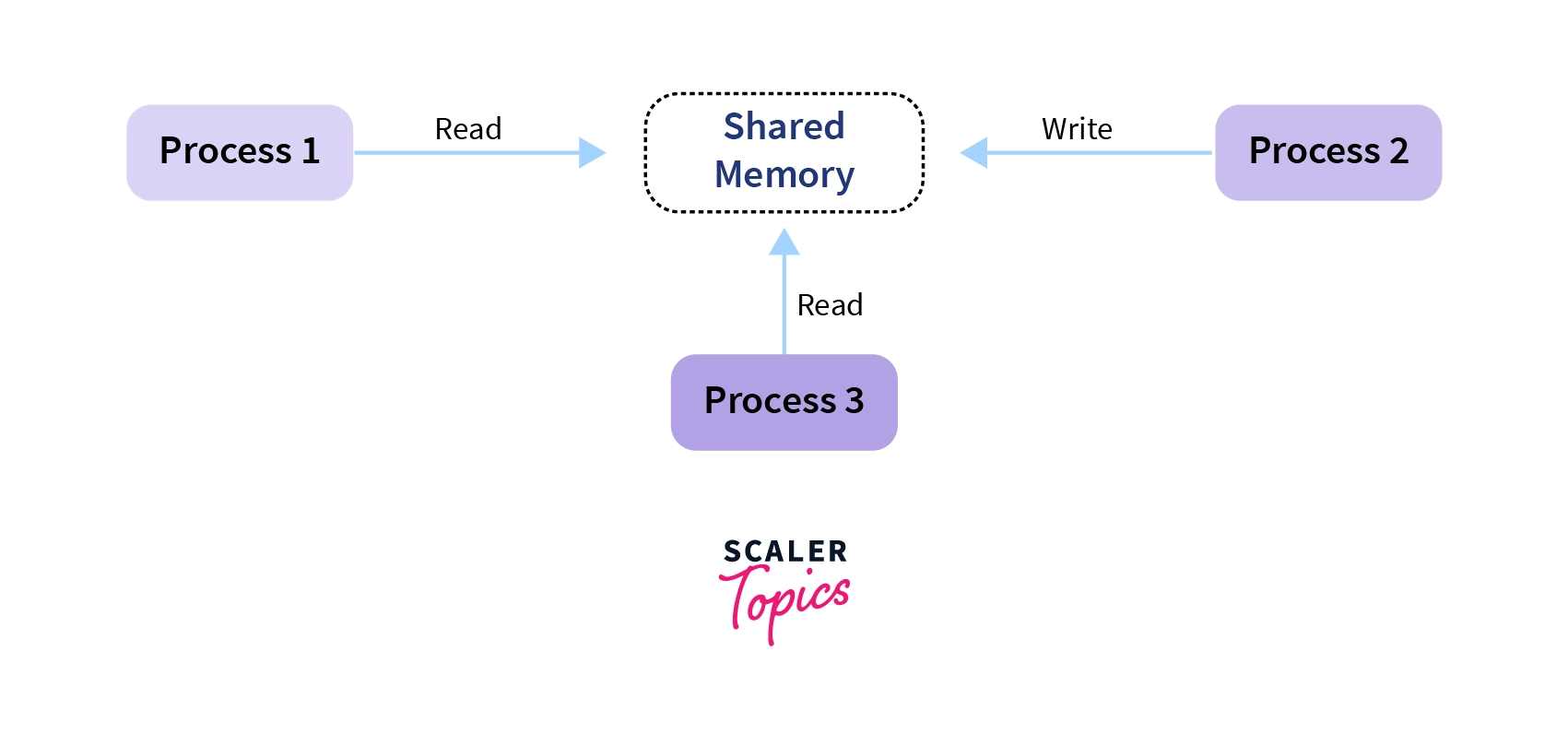


In the above image, if Process1 and Process2 happen at the same time, user 2 will get the wrong account balance as Y because of Process1 being transacted when the balance is X.

Inconsistency of data can occur when various processes share a common resource in a system which is why there is a need for process synchronization in the operating system.

**How Process Synchronization in OS Works?**

Let us take a look at why exactly we need Process Synchronization. For example, If a process1 is trying to read the data present in a memory location while another process2 is trying to change the data present at the same location, there is a high chance that the data read by the process1 will be incorrect.



Let us look at different elements/sections of a program:

* **Entry Section:** The entry Section decides the entry of a process.
* **Critical Section:** Critical section allows and makes sure that only one process is modifying the shared data.
* **Exit Section:** The entry of other processes in the shared data after the execution of one process is handled by the Exit section.
* **Remainder Section:** The remaining part of the code which is not categorized as above is contained in the Remainder section.

**Race Condition**

When more than one process is either running the same code or modifying the same memory or any shared data, there is a risk that the result or value of the shared data may be incorrect because all processes try to access and modify this shared resource. Thus, all the processes race to say that my result is correct. This condition is called the race condition. Since many processes use the same data, the results of the processes may depend on the order of their execution.

This is mostly a situation that can arise within the critical section. **In the critical section, a race condition occurs when the end result of multiple thread executions varies depending on the sequence in which the threads execute.**

But how to avoid this race condition? There is a simple solution:

* by treating the critical section as a section that can be accessed by only a single process at a time. This kind of section is called an atomic section.