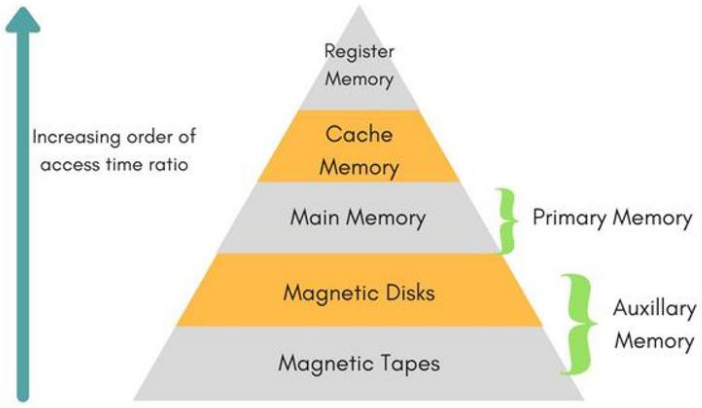
1) Describe the memory hierarchy in the computer system



The memory hierarchy in a computer system is a way of organizing computer memory to maximize performance and minimize cost. It is typically organized in layers, with each layer being faster, smaller, and more expensive than the one below it. The basic idea is to keep the

1. Registers: The fastest and smallest type of memory in a computer system, found directly on the CPU chip. Registers are used to hold data that is currently being processed by the CPU.
2. Cache memory: This is a type of high-speed memory that is built into the CPU or located near the CPU. It is used to store data that is frequently accessed by the CPU, to reduce the time it takes to access data from main memory.
3. Main memory: This is the primary storage in a computer system, typically in the form of Random Access Memory (RAM). It is used to hold data that is currently being processed by the CPU and data that is frequently accessed.
4. Secondary storage: This includes hard disk drives (HDDs), solid-state drives (SSDs), and other types of non-volatile storage. Secondary storage is slower than main memory but is used to hold data that is not currently being processed.
5. Tertiary storage: This includes archival storage devices such as tape drives and optical disks. Tertiary storage is used for long-term storage of data that is not frequently accessed.

2) Explain principle of locality of reference in detail

The principle of locality of reference is an important concept in computer architecture and memory management. It states that when a program accesses a memory location, there is a high probability that the program will access memory locations near it in the near future.

1. Spatial locality: This refers to the phenomenon where a program accesses memory locations that are near each other. For example, when a program accesses memory location x, it is likely to access memory locations x+1, x+2, and so on.
2. Temporal locality: This refers to the phenomenon where a program accesses the same memory location repeatedly over a period of time. For example, if a program is computing a sum of elements in an array, it will repeatedly access the same memory location where the array is stored.

The principle of locality of reference is important because it allows computer systems to optimize memory usage. Caching techniques, such as the use of a CPU cache or a disk cache, take advantage of locality of reference by storing frequently accessed data in faster, smaller memory locations. By doing this, the system can reduce the amount of time it takes to access data from slower, larger memory locations such as RAM or disk.

3) Explain page address translation with respect to virtual memory and further explain TLB in detail.

Page address translation is the process of converting a virtual memory address into a physical memory address. The virtual memory address is used by the program or the process running in the system, while the physical memory address is used by the system hardware to access the actual physical memory. The translation process involves the use of page tables which are maintained by the operating system.

TLB stands for Translation Lookaside Buffer, which is a hardware cache used to speed up the page address translation process. The TLB contains a subset of the page table entries and is used to store the most frequently used virtual-to-physical address mappings. When a virtual address is generated, the TLB is searched first to check whether the page is present in the cache. If it is present, the corresponding physical address is obtained directly from the TLB, which speeds up the translation process. If the page is not present in the TLB, the page table is searched as usual.

4) What is the necessity of Cache memory? Set associative cache mapping

Cache memory is a small, fast, and expensive memory that is located between the CPU and main memory. It stores frequently used instructions and data, which are fetched from main memory and placed in cache memory for faster access by the CPU. The necessity of cache memory arises due to the large difference in speed between the CPU and main memory. While the CPU can execute instructions at a very high speed, accessing main memory takes a considerable amount of time. Cache memory acts as a buffer between the CPU and main memory, reducing the average access time and improving the overall performance of the system.

Set associative cache mapping is a type of cache memory organization that allows multiple cache blocks to be stored in a single set. Each set in the cache memory contains a fixed number of cache blocks, and each block can store a fixed number of bytes of data. When a memory block is requested by the CPU, the cache controller checks if the block is present in any of the sets. If it is, the block is fetched from the cache memory and transferred to the CPU. If it is not, the block is fetched from the main memory and stored in the cache memory. Set associative cache mapping improves cache hit rates by reducing the number of conflicts that occur when multiple blocks are mapped to the same cache set.

5) Flynn's Classification

Flynn's classification is a taxonomy that categorizes computer architectures into four classes based on the number of instruction streams (IS) and the number of data streams (DS) that can be processed simultaneously. It was proposed by Michael Flynn in 1966 and is commonly used to classify modern computer systems.

1. Single Instruction, Single Data (SISD): In this category, there is a single processor that executes a single instruction on a single data item at a time. This is the simplest type of architecture and is commonly found in most traditional von Neumann computers.
2. Single Instruction, Multiple Data (SIMD): This category includes architectures that have a single instruction stream but can process multiple data streams simultaneously. Examples of SIMD architectures include vector processors and graphics processing units (GPUs).
3. Multiple Instruction, Single Data (MISD): This category includes architectures that have multiple instruction streams but only one data stream. However, there are no practical examples of this type of architecture, and it is mostly of theoretical interest.
4. Multiple Instruction, Multiple Data (MIMD): This category includes architectures that can process multiple instruction streams and data streams simultaneously. This type of architecture is commonly found in parallel computing systems, such as clusters and supercomputers.

6) Explain interrupt driven I/O method of data transfer

Interrupt-driven I/O is a method of data transfer in computer systems where the CPU initiates an I/O operation and then proceeds to execute other instructions while waiting for the I/O operation to complete. When the I/O operation is finished, the device sends an interrupt signal to the CPU, which interrupts the currently executing program and transfers control to the interrupt service routine (ISR) that handles the I/O completion.

The main advantage of interrupt-driven I/O is that it allows the CPU to perform other tasks while waiting for the completion of the I/O operation, thereby reducing the overall processing time. Interrupt-driven I/O also improves the responsiveness of the system, as the CPU can quickly switch to handle the I/O completion as soon as the device signals the completion.

7) Explain DMA Method of I/O data transfer

DMA stands for Direct Memory Access, and it is a method of transferring data between memory and peripheral devices without involving the CPU. In DMA data transfer, the data is transferred directly from the device to the memory or vice versa, without going through the CPU.

The DMA method of I/O data transfer involves a DMA controller, which is a separate hardware component that controls the data transfer between the device and the memory. When a peripheral device wants to transfer data to or from memory, it sends a request to the DMA controller. The DMA controller then takes control of the system bus and transfers the data directly between the device and memory without involving the CPU.

DMA method of I/O data transfer provides several advantages over programmed I/O, such as reduced CPU utilization, increased system throughput, and decreased latency. It is commonly used in high-speed data transfer applications, such as disk controllers, network interface cards, and graphics cards.

8) Describe memory segmentation in detail. Explain how address translation is performed in virtual memory.

Memory segmentation is a memory management technique that divides the memory into segments or sections, where each segment is a logical unit of memory. Each segment is assigned a starting address and a length, and it may contain code, data, or stack. The segment is accessed using a segment selector and an offset, where the segment selector is used to locate the segment, and the offset is used to locate the specific location within the segment.

Address translation is the process of converting a logical address into a physical address. Virtual memory is a memory management technique that allows the operating system to use more memory than is physically available by using a combination of main memory and secondary storage. In virtual memory, each process is given its own virtual address space, which is divided into pages. The operating system manages the mapping of virtual pages to physical pages.

9) State various data transfer techniques. Explain how address translation is performed in virtual memory

Various data transfer techniques are used in computer systems to transfer data between memory and peripheral devices. Some of the common techniques are:

1. Programmed I/O: In this technique, the CPU controls the data transfer between the device and memory by reading data from or writing data to the device using I/O instructions.
2. Interrupt-driven I/O: In this technique, the device initiates a request to transfer data, and the CPU responds to the request by interrupting the current task, servicing the device, and transferring data between the device and memory.
3. DMA (Direct Memory Access): In this technique, a DMA controller is used to transfer data between the device and memory directly without involving the CPU.

Address translation is a critical component of virtual memory, which is a memory management technique that allows the operating system to use more memory than is physically available. The address translation process in virtual memory involves mapping virtual addresses to physical addresses. The virtual address is divided into a page number and an offset, where the page number is used to locate the page table entry that contains the physical page number and other control bits.

The page table is a data structure that maps virtual pages to physical pages. Each page table entry contains the physical page number and control bits such as permission bits that determine the access rights of the page. The MMU (Memory Management Unit) performs the address translation by translating the virtual page number into a physical page number using the page table and adding the offset to the physical page number to generate the physical address.

The process of address translation in virtual memory is transparent to the user and applications running on the system. The virtual address space of each process is managed by the operating system, and the process can access its own virtual address space without worrying about the physical memory layout. The operating system maps the virtual pages of the process to physical pages as needed, and the MMU performs the address translation as the process accesses its virtual address space.

10) Consider a cache mapping of 16 words . Each block consists of 4 words. Size of the main memory is 256 bytes. Draw associative mapping and calculate TAG and WORD size.

11) What is mutual exclusion? Explain its significance

Mutual exclusion is a concept in computer science that refers to the ability to prevent multiple concurrent processes or threads from accessing a shared resource at the same time. This is achieved through the use of synchronization mechanisms, such as locks or semaphores, that allow only one process or thread to access the resource at a time.

The significance of mutual exclusion lies in its ability to prevent race conditions, which can occur when multiple processes or threads access a shared resource simultaneously and interfere with each other's operations. Race conditions can lead to inconsistent or erroneous results, data corruption, or even system crashes.

By enforcing mutual exclusion, concurrent processes or threads can access shared resources in a coordinated and safe manner, without interfering with each other's operations. This ensures the correctness and reliability of the system, and enables the development of complex and efficient applications that rely on concurrency.

12) Explain various file allocation techniques

In computer file systems, file allocation refers to the method of assigning storage space on a storage device to a file. There are several file allocation techniques used in computer systems, including:

1. Contiguous Allocation: In this technique, files are allocated contiguous blocks of disk space. This method is simple and easy to implement, but it suffers from fragmentation issues, where free space becomes scattered throughout the disk as files are deleted and new files are added.
2. Linked Allocation: In this technique, each file is divided into a series of blocks, and these blocks are linked together using pointers to form a linked list. Each block contains a pointer to the next block in the file. The advantage of linked allocation is that files can be of any size, and there is no fragmentation issue. However, it suffers from poor performance due to the need to follow the pointers to read or write data.
3. Indexed Allocation: In this technique, each file has an associated index block that contains pointers to all the blocks that make up the file. This index block is stored separately from the file data blocks, and its location is recorded in the file's directory entry. Indexed allocation can reduce disk fragmentation and provides fast access to files, but it requires additional disk space for the index block.
4. Multi-level Index Allocation: In this technique, an index block contains pointers to other index blocks, forming a multi-level hierarchy. This allows for efficient use of disk space while providing fast access to large files.
5. Combined Allocation: This technique combines the advantages of contiguous and linked allocation by dividing the disk into fixed-size blocks and allocating contiguous blocks to small files, while using linked allocation for larger files. This technique reduces fragmentation and provides fast access to small files, while still allowing for the allocation of large files.

13) Explain Disk Cache

Disk cache is a type of computer memory that is used to improve the performance of hard disk drives. The disk cache is a portion of random access memory (RAM) that is reserved for temporarily storing frequently accessed data from the hard disk. This data can include frequently accessed files, frequently accessed parts of files, and frequently accessed system data.

When a program requests data from the hard disk, the disk cache checks to see if the data is already stored in the cache. If the data is in the cache, it can be quickly retrieved from the cache instead of having to be read from the much slower hard disk. This can greatly improve the performance of the system, as RAM is much faster than a hard disk drive.

Overall, disk cache is an important feature that can greatly improve the performance of hard disk drives, reducing the amount of time it takes to read and write data from the disk.

14) What is deadlock? Explain the necessary sufficient condition for deadlock? What is the difference between deadlock avoidance and prevention?

Deadlock is a situation in which two or more processes are unable to proceed because each is waiting for the other to release a resource. This can occur in a system where resources are shared between multiple processes and those processes are unable to release resources in a timely manner.

The necessary conditions for deadlock are:

1. Mutual Exclusion: At least one resource must be held in a non-shareable mode. This means that only one process can access the resource at a time.
2. Hold and Wait: A process must be holding at least one resource and waiting to acquire additional resources held by other processes.
3. No Preemption: Resources cannot be preempted or taken away from a process. The resource can only be released voluntarily by the process holding it.
4. Circular Wait: A circular chain of two or more processes, each of which is waiting for a resource held by the next process in the chain.

To avoid deadlock, the system can use a resource allocation algorithm that dynamically checks resource requests and resource allocations to ensure that deadlocks cannot occur. This technique is called deadlock avoidance. It requires that the system has a prior knowledge of the resources needed by each process.

On the other hand, deadlock prevention involves modifying the system in such a way that one of the necessary conditions for deadlock is not satisfied. For example, by allowing preemption of resources or using only shareable resources. Deadlock prevention techniques are more complex and can be more difficult to implement than deadlock avoidance.

15) Explain Process synchronization in brief.

Process synchronization is a concept in operating systems that refers to the coordination of activities between multiple processes in a shared resource environment. In a multi-process system, it is important to ensure that processes execute in a safe, orderly, and predictable manner, especially when they are sharing resources such as memory or I/O devices.

Process synchronization is achieved through the use of synchronization primitives, which are mechanisms that allow processes to coordinate their activities and access shared resources in a mutually exclusive way. The most commonly used synchronization primitives are:

1. Mutexes: A mutex is a binary semaphore that is used to protect access to a shared resource. A process that wants to access the resource must acquire the mutex, which ensures that no other process can access the resource at the same time. Once the process is finished accessing the resource, it must release the mutex so that other processes can acquire it.
2. Semaphores: Semaphores are used to signal between processes or threads. A semaphore is a counter that can be incremented or decremented by processes. A process that wants to access a shared resource must first decrement the semaphore value, which indicates that the resource is in use. Once the process has finished using the resource, it must increment the semaphore value to signal to other processes that the resource is now available.
3. Condition Variables: Condition variables are used to signal between processes when a particular condition is met. A process can wait on a condition variable until another process signals it, indicating that the condition has been met.

The main goal of process synchronization is to avoid race conditions, which occur when two or more processes try to access a shared resource simultaneously and end up interfering with each other's activities. By using synchronization primitives, processes can ensure that access to shared resources is controlled and coordinated, which helps to prevent race conditions and other synchronization issues.

16) What is paging? Explain LRU, FIFO and Optimal page replacement policy for the following string. Page frame size is 4.
1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2

Paging is a memory management technique used by computer operating systems to manage the allocation of memory. It allows the operating system to allocate physical memory to running programs in smaller, fixed-size blocks called pages, rather than requiring a contiguous block of memory for each program.

The primary need for paging arises from the limited amount of physical memory available on a computer system. If a program requires more memory than is available in physical memory, it must be loaded and executed in parts. Paging provides a way to divide the program into smaller pieces, or pages, which can be loaded into physical memory as needed.

Paging also allows the operating system to manage memory more efficiently by enabling it to allocate physical memory on a per-page basis. This means that the operating system can allocate memory more flexibly, reducing the likelihood of memory fragmentation and improving overall memory utilization.

In addition, paging enables the operating system to share memory between multiple programs more efficiently. By mapping the same physical memory page to multiple virtual addresses, the operating system can reduce the amount of physical memory required to run multiple programs simultaneously.

Overall, paging is an important technique for managing memory in modern computer systems, enabling efficient use of physical memory and allowing multiple programs to run concurrently without consuming excessive amounts of memory.

17) Explain banker's algorithm in detail

The Banker's algorithm is a deadlock-avoidance algorithm used in computer operating systems. It is designed to prevent the occurrence of deadlock in systems where multiple processes compete for shared resources. The algorithm works by simulating the allocation of resources to each process and checking if the system will remain in a safe state, i.e., no deadlock will occur.

The Banker's algorithm requires the following information for each process and resource:

1. The maximum amount of each resource that the process will need during its lifetime.
2. The number of resources currently allocated to the process.
3. The number of available resources of each type in the system.

18) Reader and writer problem using semaphore

The reader-writer problem is a classic synchronization problem in computer science, where multiple processes compete for access to a shared resource. The resource in this case is a data structure, and processes can be classified as either readers or writers. Readers only want to read the data, while writers want to write to it. The problem is to ensure that no writer is allowed to access the data structure while a reader is accessing it, and vice versa.

The solution to this problem can be implemented using semaphores. A semaphore is a synchronization primitive that allows processes to control access to shared resources.

Here is one possible solution to the reader-writer problem using semaphores:

1. Define two semaphores: read_mutex and resource_mutex.
2. Initialize read_mutex to 1 and resource_mutex to 1.
3. Each reader process should perform the following steps:
   a. Wait on read_mutex semaphore.
   b. Increment the count of readers.
   c. If this is the first reader, wait on the resource_mutex semaphore.
   d. Release the read_mutex semaphore.
   e. Read the data.
   f. Wait on read_mutex semaphore.
   g. Decrement the count of readers.
   h. If this is the last reader, signal the resource_mutex semaphore.
   i. Release the read_mutex semaphore.
4. Each writer process should perform the following steps:
   a. Wait on the resource_mutex semaphore.
   b. Write to the data.
   c. Signal the resource_mutex semaphore.

The read_mutex semaphore is used to ensure that only one reader is modifying the shared counter at any given time. The resource_mutex semaphore is used to ensure that writers are not allowed to modify the shared resource while a reader is accessing it.

When a reader arrives, it first waits on the read_mutex semaphore, which prevents multiple readers from incrementing the reader count at the same time. It then increments the reader count and checks whether it is the first reader. If it is the first reader, it waits on the resource_mutex semaphore, which prevents any writer from accessing the shared resource. It then releases the read_mutex semaphore, which allows other readers to access the shared counter. After reading the data, it waits on the read_mutex semaphore, decrements the reader count, and checks whether it is the last reader. If it is the last reader, it signals the resource_mutex semaphore, which allows any waiting writer to access the shared resource.

When a writer arrives, it first waits on the resource_mutex semaphore, which prevents any other process from accessing the shared resource. It then writes to the data and signals the resource_mutex semaphore, which allows any waiting processes to access the shared resource.

This solution ensures that readers and writers do not interfere with each other, and that writers are not allowed to access the shared resource while readers are accessing it.

19) Explain Disk scheduling algorithm in detail.

Disk scheduling algorithms are used to determine the order in which the disk arm should move to access the data on the disk. The goal of these algorithms is to minimize the average access time of the disk, which includes the time required to move the disk arm to the correct track, the time required for the disk to rotate to the correct sector, and the time required to transfer the data from the disk to the memory.

Here are some of the commonly used disk scheduling algorithms:

1. FCFS (First-Come, First-Served): This algorithm services the disk requests in the order in which they arrive. It is simple to implement but can result in poor performance, especially if there are many requests that are far apart on the disk.
2. SSTF (Shortest Seek Time First): This algorithm services the disk request that requires the least amount of arm movement. It performs better than the FCFS algorithm, but can lead to starvation of requests that are far from the current position of the arm.
3. SCAN: This algorithm moves the arm in one direction, servicing all the requests in that direction until it reaches the end of the disk, and then moves the arm in the other direction, servicing all the requests in that direction. It can be more fair than SSTF and is better at reducing the average wait time, but can lead to poor performance for requests at the ends of the disk.
4. C-SCAN (Circular SCAN): This algorithm is similar to SCAN, but instead of moving the arm back to the beginning of the disk when it reaches the end, it moves the arm back to the beginning of the disk and starts servicing requests again. This can result in more uniform performance than SCAN.
5. LOOK: This algorithm is similar to SCAN, but instead of moving the arm to the end of the disk, it only moves the arm as far as the last request in its current direction. This can result in better performance than SCAN for workloads that are not uniformly distributed on the disk.
6. C-LOOK: This algorithm is similar to C-SCAN, but instead of servicing all the requests in one direction, it only services requests that are in the direction of the next request. This can result in better performance than C-SCAN for workloads that are not uniformly distributed on the disk.

To choose the best disk scheduling algorithm for a given workload, it is important to consider the workload characteristics, such as the number of requests, the distribution of requests on the disk, and the amount of time required to service each request. In general, there is no single best algorithm, and the choice of algorithm depends on the specific workload and the goals of the system.

20) Explain the effect of page frame size on performance of page replacement algorithms. 5

The page frame size, also known as page size, is an important factor that can affect the performance of page replacement algorithms in a virtual memory system. Page replacement algorithms are used to determine which pages in memory should be replaced when a new page needs to be loaded into memory. The choice of page replacement algorithm can have a significant impact on system performance, and the page frame size can affect the behavior of these algorithms in several ways.

Here are some ways in which the page frame size can affect the performance of page replacement algorithms:

1. Fragmentation: If the page frame size is too small, it can lead to external fragmentation, where there is not enough contiguous free memory to accommodate a page. This can make it more difficult for the page replacement algorithm to find a suitable page frame for new pages and increase the overhead of swapping pages in and out of memory.
2. Locality of reference: The page frame size can affect the degree of locality of reference in a program. Programs that exhibit a high degree of locality of reference tend to access memory locations that are close to each other, and a larger page frame size can improve the efficiency of the page replacement algorithm by keeping related pages in memory.
3. Thrashing: If the page frame size is too small, it can lead to thrashing, where the system spends more time swapping pages in and out of memory than executing useful work. This can result in poor system performance and slow down the execution of programs.
4. Overhead: The page frame size can also affect the overhead of the page replacement algorithm. A larger page frame size can reduce the overhead of the page replacement algorithm by reducing the number of page faults and the frequency of swapping pages in and out of memory.

In general, a larger page frame size can be beneficial in reducing fragmentation and improving the efficiency of the page replacement algorithm. However, the optimal page frame size can depend on the characteristics of the system and the workload of the programs running on it. It is important to carefully choose the page frame size and the page replacement algorithm to ensure optimal system performance.

21) Explain Thrashing.

Thrashing is a phenomenon in computer systems that occurs when the system is spending too much time and resources on paging and swapping, rather than on executing useful work. Thrashing occurs when the system is under heavy load and does not have enough physical memory to accommodate all the processes that are running.

When the system runs out of physical memory, it begins to swap pages of memory to disk. This is done to free up physical memory so that it can be used for other processes. However, if the system is swapping pages of memory to disk at a high rate, then it is likely that the system is thrashing.

Thrashing is a serious problem because it can cause a significant decrease in system performance. When the system is thrashing, the disk is being used heavily for swapping pages of memory, which can cause a bottleneck in the system. As a result, the system spends more time swapping pages of memory than executing useful work, which can cause the system to become unresponsive or slow.

To prevent thrashing, it is important to ensure that the system has sufficient physical memory to accommodate all the processes that are running. It is also important to use an efficient page replacement algorithm that can quickly identify and swap out pages of memory that are not being used. Finally, it is important to limit the number of processes that are running on the system to avoid overloading the system.

22) Consider the following snapshot of the system. Using Bankers Algorithm, determine whether or not system is in safe state. If yes determine the safe sequence.

Consider the following snapshot of the system. Using Bankers Algorithm, determine whether or not system is in safe state. If yes determine the safe sequence.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	3	0	1	4	5	1	1	7	0	3	0	1
P1	2	2	1	0	3	2	1	1				
P2	3	1	2	1	3	3	2	1				
P3	0	5	1	0	4	6	1	2				
P4	4	2	1	2	6	3	2	5				

23) If requests for p3 (0,1,0,1) can be satisfied

Calculate number of page faults and page hits for the page replacement policies FIFO, Optimal and LRU for given reference string 6, 0, 5, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 5, 2, 0, 5, 6, 0, 5 (assuming three frame size).
Explain synchronization problem in detail. How counting semaphore can be used to solve readers writers problem.

Given memory partitions of 150k,500k,200k,300k,550k(in order) how would each of the first fit, best fit and worst fit algorithm places the processes of 220k,430k,110k,425k(in order).Evaluate, which algorithm makes most efficient use of memory?

Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO is ordered as 80, 1470, 913, 1777, 948, 1022, 1750,130. What is the total distance that the disk arm moves for following by applying following algorithms?
1. FCFS 2. SSTF 3. LOOK 4. SCAN

24)

1. FCFS (First-Come-First-Serve):

- Total distance = 10714 cylinders
- Calculation:
  - Current position: 143
  - Distance to 86: 57 cylinders (143-86)
  - Distance to 1470: 1327 cylinders (1470-143)
  - Distance to 913: 557 cylinders (1470-913)
  - Distance to 1774: 861 cylinders (1774-913)
  - Distance to 948: 826 cylinders (1774-948)
  - Distance to 1509: 561 cylinders (1509-948)
  - Distance to 1022: 487 cylinders (1509-1022)
  - Distance to 1750: 728 cylinders (1750-1022)
  - Distance to 130: 1620 cylinders (1750-130)
  - Total distance: 57+1327+557+861+826+561+487+728+1620 = 10714 cylinders

2. SSTF (Shortest Seek Time First):

- Total distance = 2364 cylinders
- Calculation:
  - Current position: 143
  - Distance to 130: 13 cylinders (143-130)
  - Distance to 86: 57 cylinders (86-130)
  - Distance to 1774: 1631 cylinders (1774-143)
  - Distance to 1750: 24 cylinders (1774-1750)
  - Distance to 1509: 241 cylinders (1509-1750)
  - Distance to 1470: 39 cylinders (1509-1470)
  - Distance to 1022: 448 cylinders (1022-1470)
  - Distance to 948: 74 cylinders (1022-948)
  - Distance to 913: 35 cylinders (948-913)
  - Total distance: 13+57+1631+24+241+39+448+74+35 = 2364 cylinders

3. SCAN (Elevator):

- Total distance = 2820 cylinders
- Calculation:
  - Current position: 143
  - Distance to 0: 143 cylinders (143-0)
  - Distance to 86: 57 cylinders (86-0)
  - Distance to 130: 44 cylinders (130-86)
  - Distance to 1774: 1644 cylinders (1774-130)
  - Distance to 1750: 24 cylinders (1774-1750)
  - Distance to 1509: 241 cylinders (1774-1509)
  - Distance to 1470: 39 cylinders (1470-1509)
  - Distance to 1022: 448 cylinders (1509-1022)
  - Distance to 948: 74 cylinders (948-1022)
  - Distance to 913: 35 cylinders (913-948)
  - Distance to 1750: 837 cylinders (1750-913)
  - Total distance: 143+57+44+1644+24+241+39+448+74+35+837 = 2820 cylinders

4. LOOK:

- Total distance = 2236 cylinders
- Calculation:
  - Current position: 143
  - Distance to 86: 57 cylinders (143-86)
  - Distance to 130: 44 cylinders (130-86)
  - Distance to 1774: 1644 cylinders (1774-130)
  - Distance to 1750: 24 cylinders (1774-1750)

25) Explain counting semaphore with examples

A counting semaphore is a synchronization mechanism that allows multiple threads or processes to access a shared resource concurrently, but with a certain limit. It maintains an integer value, which is used to represent the number of available units of a resource.

When a thread or process wants to access the resource, it checks the value of the semaphore. If the value is greater than zero, it decrements the value of the semaphore and proceeds with accessing the resource. If the value is zero, the thread or process blocks until the value of the semaphore becomes greater than zero.

For example, suppose there are 6 threads trying to print documents on the printer simultaneously. The first 5 threads will be able to access the printer immediately, as the semaphore value will be decremented from 5 to 0. The 6th thread will be blocked until one of the previous threads finishes printing and increments the semaphore value back to 1, allowing the 6th thread to access the printer.

In this way, the counting semaphore ensures that no more than 5 print jobs are processed simultaneously, while allowing multiple threads to access the printer concurrently.

26) Explain data structure used in banker's algorithm

1. Available Vector: A one-dimensional array of size m, where m is the number of resource types in the system. It contains the number of available resources of each type.
2. Allocation Matrix: A two-dimensional matrix of size n x m, where n is the number of processes in the system and m is the number of resource types in the system. It contains the number of resources of each type allocated to each process.
3. Need Matrix: A two-dimensional matrix of size n x m, where n is the number of processes in the system and m is the number of resource types in the system. It contains the number of resources of each type that each process still needs to complete its task.
4. Request Vector: A one-dimensional array of size m, where m is the number of resource types in the system. It contains the number of resources of each type that a process is requesting.

27) Explain virtual memory concept with respect to paging

Virtual memory is a technique used by operating systems to allow a computer to use more memory than is physically available. It does this by using a portion of the computer's hard disk as an extension of its RAM.

The virtual memory concept with respect to paging involves breaking down a program's memory into small fixed-size blocks called pages. The size of the pages is typically 4KB or 8KB. Each page is mapped to a frame in physical memory. The mapping between pages and frames is maintained by the operating system's memory management unit.

When a program needs to access memory, it generates a virtual address that consists of a page number and an offset within the page. The memory management unit translates this virtual address into a physical address that specifies the frame number and the offset within the frame.

If a program attempts to access a page that is not currently in physical memory, a page fault occurs. The operating system then selects a page to evict from physical memory and swaps it out to disk to make room for the requested page. This process is known as paging.

Paging allows programs to access more memory than is physically available by swapping pages in and out of physical memory as needed. It also allows multiple programs to share the same physical memory without interfering with each other.

In summary, virtual memory with respect to paging is a technique used by operating systems to allow programs to use more memory than is physically available. It does this by breaking down a program's memory into pages and swapping pages in and out of physical memory as needed.

28) Explain Resource Allocation graph

Resource Allocation Graph is a graphical tool used to detect deadlocks in a system that uses shared resources. It represents the resources and processes in the system as nodes and edges respectively. There are two types of nodes in a resource allocation graph: resource nodes and process nodes. Resource nodes represent resources that can be shared by multiple processes, while process nodes represent the individual processes that require resources to execute.

The edges between nodes in the graph represent requests and allocations of resources. An arrow from a process node to a resource node represents a request for the resource by the process, while an arrow from a resource node to a process node represents the allocation of the resource to the process. A cycle in the graph indicates the possibility of a deadlock.

To analyze the graph, we look for cycles that involve only request edges. If we find such a cycle, then it indicates that the processes involved in the cycle are deadlocked. In order to break the deadlock, we must select one or more processes in the cycle and force them to release some of their allocated resources, which can then be used by other processes to proceed with their execution.

29) What are Semaphores? Differentiate between Counting and Binary Semaphores.

Semaphores are a synchronization primitive used in concurrent programming to control access to shared resources. They are essentially integer variables that are used to indicate the status of a shared resource or the availability of a shared resource.

Counting Semaphores	Binary Semaphores
Can take on any non-negative value	Can only take on 0 or 1
Used to control access to a pool of resources	Used to control access to a single resource
Allow multiple processes to access the resource	Only one process can access the resource at a time
When a process acquires the semaphore, the value is decremented	When a process acquires the semaphore, the value is set to 0
When a process releases the semaphore, the value is incremented	When a process releases the semaphore, the value is set to 1

30) Discuss Dinning Philosopher problem.

Binary semaphore is a synchronization mechanism that allows only one thread or process to access a shared resource at a time. It maintains a binary value, which is used to represent the availability of a resource.

When a thread or process wants to access the resource, it checks the value of the semaphore. If the value is 1, it decrements the value of the semaphore to 0 and proceeds with accessing the resource. If the value is 0, the thread or process blocks until the value of the semaphore becomes 1.

The Dining Philosophers problem is a classic synchronization problem that illustrates the challenges of coordinating access to shared resources among multiple processes or threads. The problem consists of five philosophers sitting at a round table, each with a plate of spaghetti and a fork to the left and right of them. The philosophers spend their time thinking and eating. When a philosopher wants to eat, they need to use both forks on either side of them. However, only one philosopher can use a fork at a time.

The challenge is to design a synchronization protocol that allows the philosophers to eat without deadlock or starvation. One solution to this problem involves using binary semaphores to control access to the forks. Each fork can be represented by a binary semaphore, and a philosopher can only eat if they can acquire both forks on either side of them.

However, this solution can lead to deadlock if all philosophers pick up their left forks at the same time and then wait indefinitely for the right fork to become available. To avoid this situation, a simple modification to the solution involves introducing a limit on the number of philosophers that can pick up their forks at the same time.

For example, we can use a binary semaphore that represents the number of available forks. Initially, the semaphore value is set to 4, indicating that 4 forks are available. When a philosopher wants to eat, they first check the value of the semaphore. If the value is greater than 1, they acquire both forks and proceed with eating. If the value is 1 or 0, they release their left fork and try again later.

In this way, the use of binary semaphores can help prevent deadlock in the Dining Philosophers problem by controlling access to shared resources and ensuring that all threads or processes can make progress towards their goals.

31) What do you understand by a deadlock? Explain deadlock avoidance method.

A deadlock occurs in a computer system when two or more processes are unable to proceed because each is waiting for one or more of the others to complete some action, resulting in a situation where no progress can be made.

Deadlock avoidance is a method of preventing deadlocks from occurring in a computer system. It involves analyzing the potential interactions between processes and the resources they require to ensure that deadlock cannot occur. There are several techniques for deadlock avoidance:

1. Resource allocation graph: A resource allocation graph can be used to determine whether a system is in a safe state. If it is not, then deadlock is possible. To avoid deadlock, resources should be allocated in such a way that the graph is always in a safe state.
2. Banker's algorithm: The banker's algorithm is a resource allocation algorithm that is used to avoid deadlock. It works by requiring processes to declare their maximum resource requirements at the beginning of their execution. The system can then use this information to determine whether granting a request will result in deadlock.

32) Explain different types of memory fragmentation

Memory fragmentation occurs when the available memory in a system is broken up into smaller, non-contiguous blocks. This can happen over time as memory is allocated and deallocated by programs, leading to fragmentation of the memory space. There are two main types of memory fragmentation:

1. Internal Fragmentation: This type of fragmentation occurs when a process requests a block of memory and is allocated more memory than it actually needs. The extra space is wasted, resulting in a loss of available memory. Internal fragmentation can be minimized by allocating memory in fixed-size blocks, or by using memory allocation techniques that allow for more precise control over the size of allocated memory.
2. External Fragmentation: This type of fragmentation occurs when the total amount of free memory in the system is sufficient to satisfy a request for memory, but the memory is divided into smaller blocks that are not contiguous. This results in a situation where the requested memory cannot be allocated, even though there is enough free memory available. External fragmentation can be minimized by using memory allocation techniques that compact free memory blocks into larger contiguous blocks.

33) Compare the performance of FIFO, LRU and Optimal based on number of page hit for the following string. Frame size =3; String (pages): 1 2 3 4 5 2 1 3 3 2 4 5.

34) Explain Interrupt driven IO and discuss the advantages of Interrupt driven IO over programmed IO.

In computing, input/output (IO) operations refer to the communication between a computer's central processing unit (CPU) and external devices such as disks, keyboards, printers, and network cards. IO can be performed either through programmed IO or interrupt-driven IO.

Programmed IO involves the CPU executing instructions to transfer data between memory and IO devices. In this approach, the CPU initiates the IO operation, waits for the IO device to complete the operation, and then continues with other tasks. This approach is straightforward but can be inefficient as it ties up the CPU, preventing it from doing other useful work while waiting for IO operations to complete.

In contrast, interrupt-driven IO is a more efficient approach that allows the CPU to perform other tasks while waiting for IO operations to complete. In this approach, the IO device interrupts the CPU when it is ready to transfer data. The CPU then services the interrupt, which involves temporarily suspending the current process, executing an interrupt service routine (ISR), and transferring data between the IO device and memory.

Interrupt-driven IO offers several advantages over programmed IO:

1. Improved CPU utilization: Interrupt-driven IO allows the CPU to perform other tasks while waiting for IO operations to complete, improving CPU utilization and overall system performance.
2. Reduced latency: With interrupt-driven IO, IO operations can be initiated and completed more quickly, reducing the overall latency of IO operations.
3. Reduced CPU overhead: Interrupt-driven IO reduces the amount of CPU overhead required for IO operations, allowing the CPU to perform other tasks more efficiently.
4. Scalability: Interrupt-driven IO can scale to support a large number of IO devices and concurrent IO operations.

However, interrupt-driven IO also has some drawbacks, such as increased complexity and overhead associated with handling interrupts and interrupt service routines. Nonetheless, interrupt-driven IO is generally considered to be a superior approach to programmed IO in terms of efficiency, performance, and scalability.

35) Discuss various File Allocation Mechanism and their advantages.

1. Contiguous allocation: This mechanism allocates contiguous blocks of disk space to a file. The file is allocated a starting block and a length, and all the blocks between the starting block and the last block are allocated to the file. This method is simple and efficient, but it suffers from external fragmentation. Files may become fragmented over time as they are modified and resized.

Advantages:

- Simple and efficient mechanism
- Fast access to files since the blocks are contiguous

2. Linked allocation: This mechanism allocates non-contiguous blocks of disk space to a file. Each block contains a pointer to the next block in the file. The last block has a null pointer to signify the end of the file. This method avoids external fragmentation, but it suffers from internal fragmentation.

Advantages:

- Efficient use of disk space
- No external fragmentation

3. Indexed allocation: This mechanism uses a separate index block to keep track of the blocks allocated to a file. The index block contains pointers to the actual blocks on the disk. This method allows for random access to files and avoids fragmentation, but it requires additional disk space to store the index block.

Advantages:

- Efficient use of disk space
- Random access to files
- No fragmentation

36) A counting semaphore was initialized to 8. Then 5 wait () operations and 2signal () operations were completed on this semaphore. What is the resulting value of the semaphore?

If a counting semaphore was initialized to 8, then its initial value would be 8. When 5 wait() operations are completed, the value of the semaphore would be decremented by 5, since each wait() operation decrements the value by 1. Therefore, the resulting value of the semaphore after 5 wait() operations would be:

8 - 5 = 3

After 2 signal() operations are completed, the value of the semaphore would be incremented by 2, since each signal() operation increments the value by 1. Therefore, the resulting value of the semaphore after 2 signal() operations would be:

3 + 2 = 5

So the resulting value of the semaphore would be 5 after 5 wait() operations and 2 signal() operations.

37) Hardware Multithreading

Hardware multithreading, also known as simultaneous multithreading (SMT), is a technique used in computer processors to improve performance by executing multiple threads in parallel. In hardware multithreading, a processor can execute multiple threads from different processes simultaneously, allowing for more efficient use of its resources.

Hardware multithreading works by dividing the processor into multiple logical processors, or threads. Each thread has its own set of registers and program counter, allowing it to execute independent instructions. These threads share the processor's functional units, such as the arithmetic logic unit (ALU) and the floating-point unit (FPU), allowing them to execute instructions in parallel.

There are two main types of hardware multithreading: fine-grained multithreading and coarse-grained multithreading. In fine-grained multithreading, the processor switches between threads on a cycle-by-cycle basis, allowing instructions from different threads to be executed in the same clock cycle. In coarse-grained multithreading, the processor switches between threads less frequently, such as when a thread is blocked waiting for memory access or input/output (I/O) operations.

Hardware multithreading can improve performance by allowing a processor to execute more instructions in parallel, reducing the amount of idle time and increasing throughput. However, it also requires additional hardware resources, such as additional registers and logic for thread switching, and may increase power consumption. Additionally, some applications may not be well-suited for hardware multithreading if they do not have enough parallelism to take advantage of the additional threads.

Consider the following pseudocode where S is a semaphore initialised to 5 in line #2 and counter is a shared variable initialised to 0 in line #1. Assume that the increment operation in line #7 in not atomic.

```
int counter = 0;
semaphore S = init(5);
void parop(void) {
  wait(5);
  wait(5);
  counter++;
  signal(5);
  signal(5);
}
```

if five threads execute the function parop concurrently which of the following program behavior is/are possible?
1. The value of counter is 5 after all the threads successfully complete the execution of parop.
2. The value of counter is 1 after all the threads successfully complete the execution of parop.
3. The value of counter is 0 after all the threads successfully complete the execution of parop.
4. There is a deadlock involving all the threads.

The possible values of the counter after all threads successfully complete the execution of the `parop` function are 1 and 5.

- The semaphore `S` is initialized to 5, which means that up to 5 threads can execute the `wait(5)` operation without being blocked. Once the value of `S` reaches 0, any subsequent thread trying to execute the `wait(5)` operation will be blocked until another thread executes the `signal(5)` operation to increment the value of `S`.
- In the `parop` function, each thread executes two `wait(5)` operations, which means that the first 5 threads will be able to execute these operations without being blocked. However, once the 6th thread tries to execute the `wait(5)` operation, it will be blocked until another thread executes the `signal(5)` operation.
- The `counter++` operation is not atomic, which means that it consists of multiple machine instructions that can be interrupted by other threads. This can lead to race conditions and incorrect results if multiple threads try to modify the value of `counter` at the same time.
- If all 5 threads successfully execute the `parop` function, then they will execute the `signal(5)` operation twice each, which means that the value of `S` will be incremented by 10, and the value of `counter` will be incremented by 5.
- If any of the threads is blocked by the `wait(5)` operation, then the value of `S` will be less than 5, and the other threads will not be able to execute the `wait(5)` operation, which can lead to a deadlock if all threads are blocked.

Therefore, options 1 and 2 are possible, and options 3 and 4 are not possible.

Given page reference string: ● 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6 ● Compare the number of page faults for LRU, FIFO and Optimal page replacement algorithm

A computer has twenty physical page frames which contain pages numbered 101 through 120. Now a program accesses the pages numbered 1, 2, …, 100 in that order, and repeats the access sequence THRICE. Which one of the following page replacement policies experiences the same number of page faults as the optimal page replacement policy for this program?
(A) Least-recently-used
(B) First-in-first-out
(C) Last-in-first-out
(D) Most-recently-used

Answer: (D)

Consider a storage disk with 4 platters (numbered as 0, 1, 2 and 3), 200 cylinders (numbered as 0, 1, … , 199), and 256 sectors per track (numbered as 0, 1, … 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:

[120, 72, 2], [180, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 16, 1]

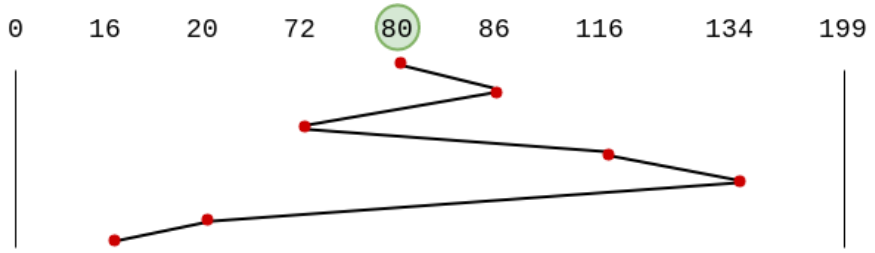
Currently head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts. Power dissipation associated with rotational latency and switching of head between different platters is negligible.

The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithm is _____ .

Note –This was Numerical Type question.
(A) 45
(B) 80
(C) 85
(D) None of these

Answer: (C)

Explanation: Head starts at 80.



Total Head movements in SSTF = (86-80) + (86-72) + (134-72) + (134-16) = 200

Power dissipated by 200 movements : P1 = 0.2 * 200 = 40 mW
Power dissipated in reversing head direction once = 15 mW
Number of time Head changes its direction = 3

Power dissipated in reversing head direction: P2 = 3 * 15 = 45 mW
Total power consumption (in mW) is P1 + P2 = 40 mW + 45 mW = 85 mW

So, answer is 85.

Consider the following five disk five disk access requests of the form (request id, cylinder number) that are present in the disk scheduler queue at a given time.

$(P, 155), (Q, 85), (R, 110), (S, 30), (T, 115)$

Assume the head is positioned at cylinder 100. The scheduler follows Shortest Seek Time First scheduling to service the requests.

Which one of the following statements is FALSE?

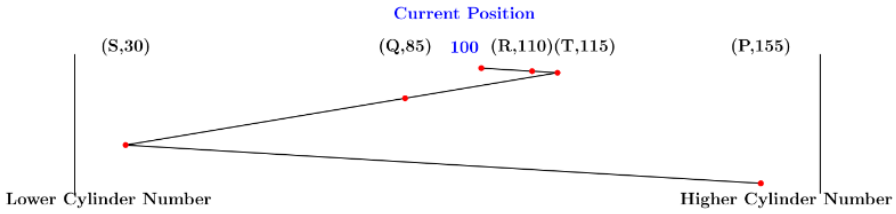
A. T is serviced before P.
B. Q is serviced after S,but before T.
C. The head reverses its direction of movement between servicing of Q and P.
D. R is serviced before P.

Ans:

Shortest Seek Time First (SSTF), selects the request with minimum to seek time first from the current head position.
In the given question disk requests are given in the form of $\langle request\ id,\ cylinder\ number \rangle$

Cylinder Queue: $(P, 155), (Q, 85), (R, 110), (S, 30), (T, 115)$

Head starts at: 100



- It is clear that R and T are serviced before P.
- Q is serviced when head is moving towards lower cylinders and P is serviced when head is moving towards higher cylinders thus reverses it direction at S.

Option B) is the correct answer

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Consider the following five disk five disk access requests of the form (request id, cylinder number) that are present in the disk scheduler queue at a given time.

(P,155),(Q,85),(R,110),(S,30),(T,115)

Assume the head is positioned at cylinder 100. The scheduler follows Shortest Seek Time First scheduling to service the requests.

Which one of the following statements is FALSE?

A. T is serviced before P.

B. Q is serviced after S, but before T.

C. The head reverses its direction of movement between servicing of Q and P.

D. R is serviced before P.

Consider the following multi-threaded code segment (in a mix of C and pseudo-code), invoked by two processes P_1 and P_2, and each of the processes spawns two threads T_1 and T_2:

```
int x = 0;  // global
Lock L1;    // global
main () {
    create a thread to execute foo(); // Thread T1
    create a thread to execute foo(); // Thread T2
    wait for the two threads to finish execution;
    print(x);}


foo() {
    int y = 0;
    Acquire L1;
    x = x + 1;
    y = y + 1;
    Release L1;
    print (y);}
```

Which of the following statement(s) is/are correct?

A. Both P_1 and P_2 will print the value of x as 2.
B. At least of P_1 and P_2 will print the value of x as 4.
C. At least one of the threads will print the value of y as 2.
D. Both T_1 and T_2, in both the processes, will print the value of y as 1.

ANS –

Each process has its own address space.

1. P_1 :
   Two threads T_{11}, T_{12} are created in main.

   Both execute foo function and threads don't wait for each other. Due to explicit locking mechanism here mutual exclusion is there and hence no race condition inside foo().

   y being thread local, both the threads will print the value of y as 1.

   Due to the wait in main, the print(x) will happen only after both the threads finish. So, x will have become 2.

   PS: Even if x was not assigned 0 explicitly in C all global and static variables are initialized to 0 value.

2. P_2 :

   Same thing happens here as P_1 as this is a different process. For sharing data among different processes mechanisms like shared memory, files, sockets etc must be used.

So, the correct answer here is A and D.

- Suppose wait is removed from the main(). Then the possible x values can be $0, 1, 2$ as the main thread as well as the two created threads can execute in any order.
- Suppose locking mechanism is removed from foo() and assignments are not atomic. (If increment is atomic here, then locking is not required). Then race condition can happen and so one of the increments can overwrite the other. So, in main, x value printed can be either 1 or 2.
- Now suppose we had just one process which does a fork() inside main before creating the threads. How the answer should change?

TT - 2 Syllabus:

Q1 Deadlock - 7

Q2 File and I/O management - 8

Q3 Processor Architectures - 5

Q4 Deadlock – 5

Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

   Work* = *Available

   ***Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both:

   (a) Finish* [*i*] = *false

   ***(b)* $Need_i \leq Work$**

   If no such ***i*** exists, go to step 4

3. ***Work* = *Work* +**
   $Allocation_i$ ***Finish*[*i*] = *true***
   go to step 2

4. If ***Finish* [*i*] == *true*** for all ***i***, then the system is in a safe state

Deadlock Detection Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:

   (a) Work = Available

   For ***i* = 1,2, …, *n***, if $Allocation_i \neq 0$, then
   Finish*[i] = *false; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:

   (a) Finish*[*i*] == *false

   ***(b)* $Request_i \leq Work$**

   (c) If no such ***i*** exists, go to step 4

3. ***Work* = *Work* +** $Allocation_i$
   Finish*[*i*] = *true
   go to step 2

4. If ***Finish[i] == false***, for some ***i***, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if ***Finish*[*i*] == *false***, then P_i is deadlocked

Algorithm requires an order of O(m x n^2) operations to detect whether the system is in deadlocked state