MODEL ANSWER KEY

QP CODE: 40533 SUB: OS SEM: IV CBCGS

Q.1. a) What are the major activities of an Operating system with regard to file management and memory management? 10M

b) Compare and contrast stateless and stateful service with the help of an example. 10M

Q.2. a) Explain with the help of an example, which of the following scheduling algorithms could result in starvation? 10M

a. First-come, first-served

b. Shortest job first

c. Round robin

d. Priority

b) What resources are used when a thread is created? How do they differ from those used when a process is created? 10M

Q.3. a) Show that, if the wait () and signal () semaphore operations are not executed atomically, then mutual exclusion may be violated. 10M

b) Consider the following snapshot of a system: 10M

	Allocation	*Max*	*Available*
	ABCD	*ABCD*	*ABCD*
Po	0012	0012	1520
pl	1000	1750	
p2	1354	2356	
p3	0632	0652	
p4	0014	0656	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix *Need?*

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

Q.4. a) With the help of a neat labeled diagram, explain the hardware support with TLB for paging. 10M

b) Consider the following page reference string: 10M

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

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, and seven frames?

Remember that all frames are initially empty, so your first unique pages will cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

Q.5. a) Justify the statement: Demand paging can significantly affect the performance of computer system. 10M

b) Compare and contrast given allocation methods: Contiguous allocation, Linked allocation, Indexed allocation. 10M

Q.6. Write Short Notes on: (Any four) 20M

a) Just-in-time compiler.

b) Memory segmentation

c) Deadlock avoidance in distributed system.

d) Operating System Schedulers

e) File system organization

f) Two-phase locking protocol

Q.1 a)

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.

Decides who gets the resources.

Allocates the resources.

De-allocates the resources.

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.

In multiprogramming, the OS decides which process will get memory when and how much.

Allocates the memory when a process requests it to do so.

De-allocates the memory when a process no longer needs it or has been terminated.

b)

Stateless Server

- requests are self-contained (every access may need translation)
- better fault tolerance
- open/close at client (fewer msgs)
- no space reserved for file descriptor tables
- thus, no limit of open files
- no problem if client crashes

Stateful Servers

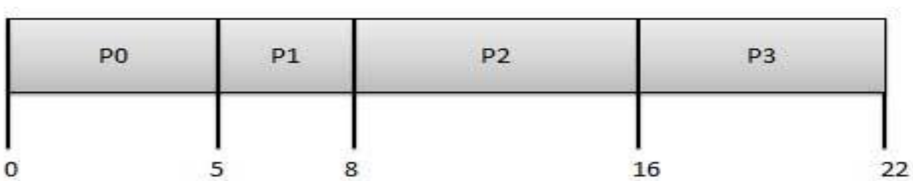
- shorter messages
- better performance (info in memory until close)
- open/close at server
- file locking possible
- read ahead possible

Q.2 a)

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

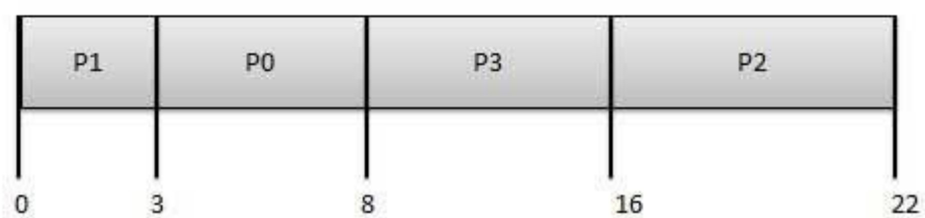
Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Wait time of each process is as follows −

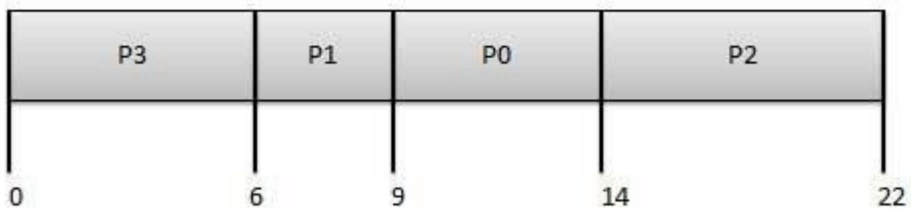
Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

Average Wait Time: (3+0+14+5) / 4 = 5.50

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows −

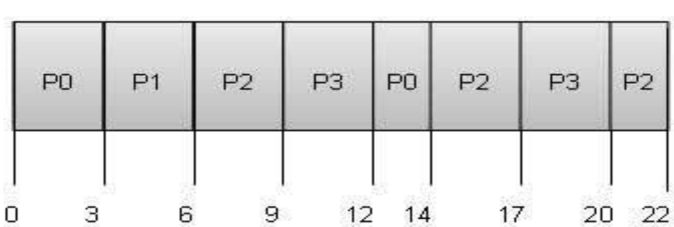
Process	**Wait Time : Service Time - Arrival Time**
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Quantum = 3



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

b) When a thread is created the threads does not require any new resources to execute the thread shares the resources like memory of the process to which they belong to. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.Where as if a new process creation is very heavyweight because it always requires new address space to be created and even if they share the memory then the inter process communication is expensive when compared to the communication between the threads.

Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

Q.3 a) " Semaphore S is an integer variable that is accessed through standard **atomic** operations i.e. wait() and signal().

It also provided basic definition of wait()

```
wait(Semaphore S)
{
  while S<=0
    ; //no operation
  S--;
}
```

Definition of signal()

```
signal(S)
{
  S++;
}
```

Let the initial value of a semaphore be 1, and say there are two concurrent processes P0 and P1 which are not supposed to perform operations of their critical section simultaneously.

Now say P0 is in its critical section, so the Semaphore S must have value 0, now say P1 wants to enter its critical section so it executes wait(), and in wait() it continuously loops, now to exit from the loop the semaphore value must be incremented, but it may not be possible because according the source, wait() is an atomic operation and can't be interrupted and thus the process P0 can't call signal() in a single processor system.

Q.3 b) Need Matrix

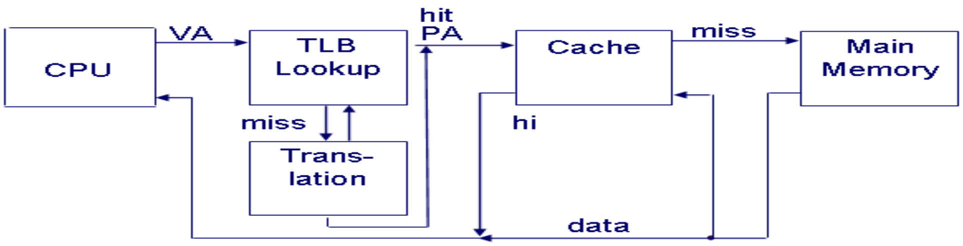
0	0	0	0
0	7	5	0
1	0	0	2
0	0	2	0
0	6	4	2

<P0, P2, P3, P4, P1>

Request from P1 can be granted immediately.

Q.4 a)

Since the page tables are stored in the main memory, each memory access of a program requires at least one memory accesses to translate virtual into physical address and to try to satisfy it from the cache. On the cache miss, there will be two memory accesses. The key to improving access performance is to rely on locality of references to page table. When a translation for a virtual page is used, it will probably be needed again in the near future because the references to the words on that page have both temporal and spatial locality.



Each virtual memory reference can cause two physical memory accesses :
-One to fetch the page table.
-One to fetch the data.

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, processor examines the TLB If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table. TLB first checks if page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.

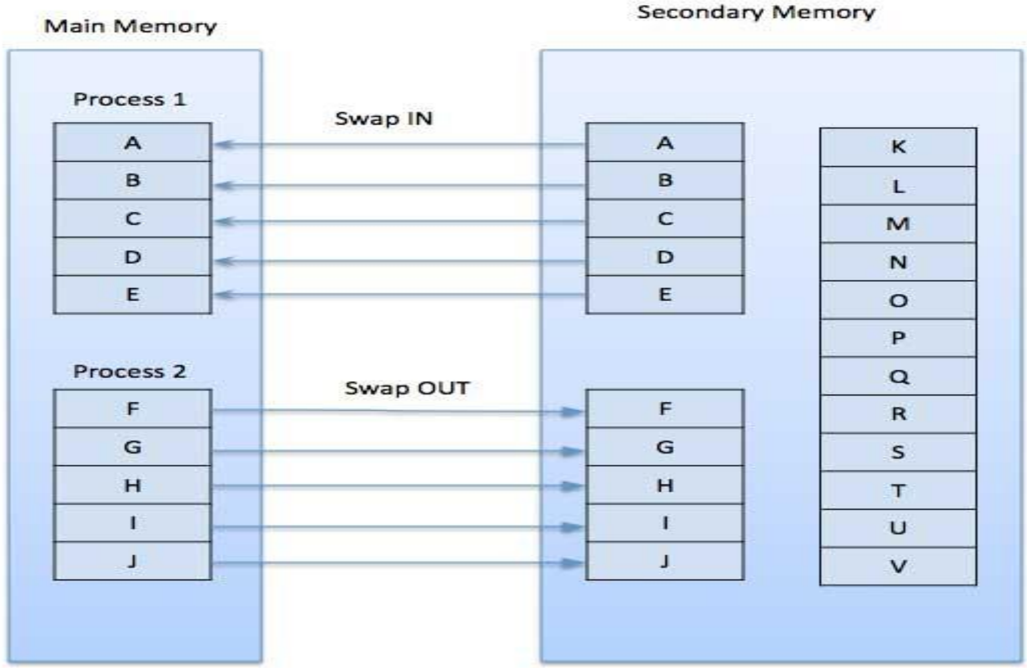
Q.4 b)

Frames	F=1		F=2		F=3		F=4		F=5		F=6		F=7	
Algo	H	M	H	M	H	M	H	M	H	M	H	M	H	M
FIFO	0	20	2	18	4	16	6	14	10	10	10	10	13	7
LRU	0	20	2	18	5	15	10	10	12	8	12	8	13	7
OPT	0	20	5	15	9	11	12	8	13	7	13	7	13	7

Q.5 a)

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

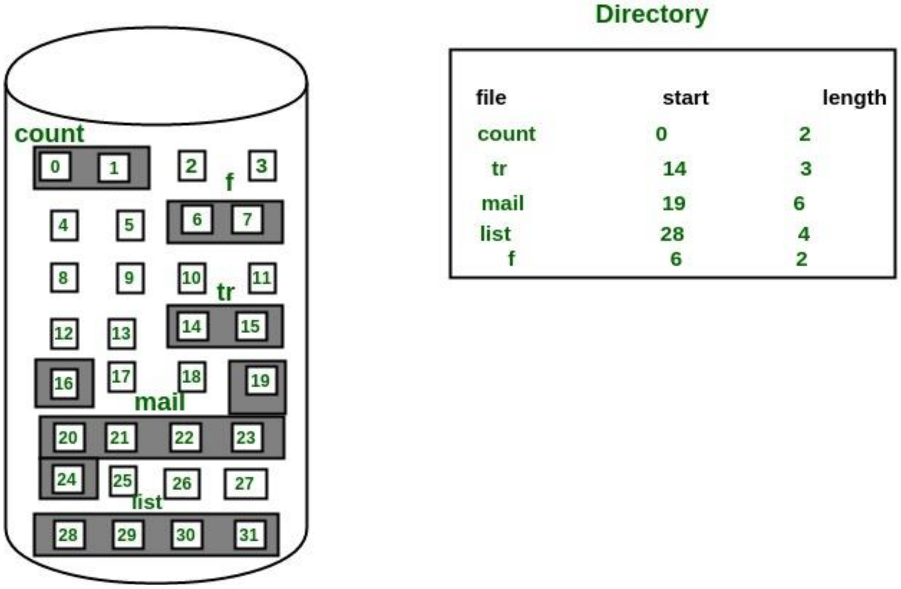
b) **Contiguous Allocation**

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1*. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

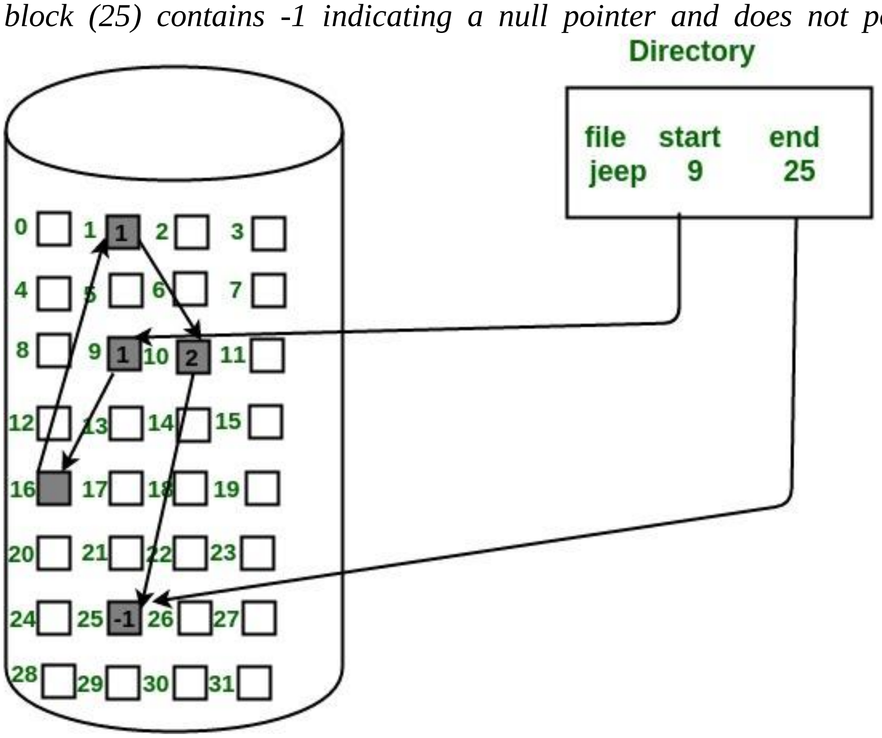
Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation

In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk.
The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

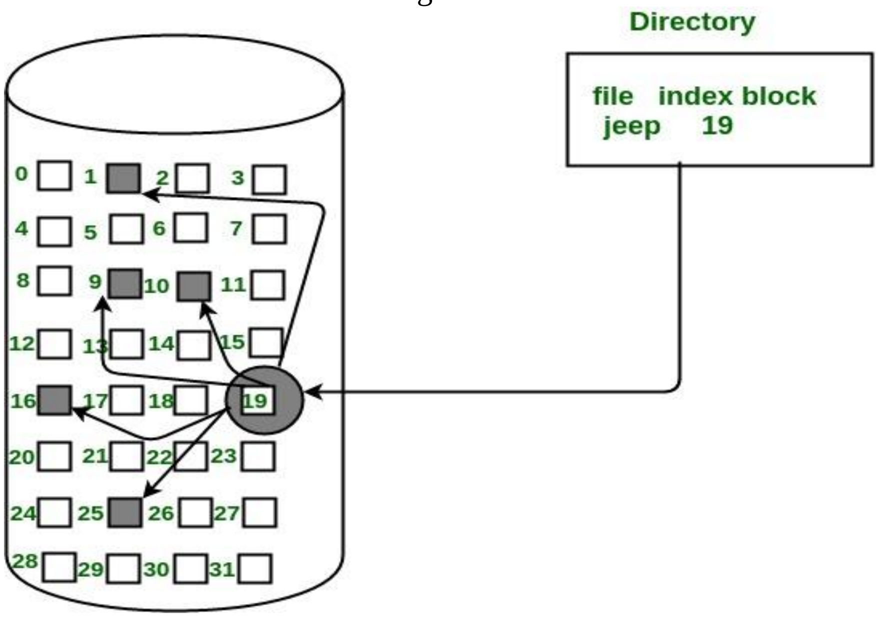
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

3. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Q.6 a) The Just-In-Time (JIT) compiler is a component of the Java™ Runtime Environment that improves the performance of Java applications at run time.

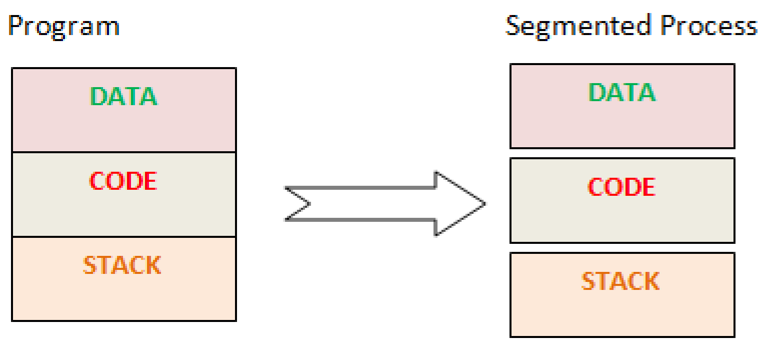
Java programs consists of classes, which contain platform-neutral bytecodes that can be interpreted by a JVM on many different computer architectures. At run time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation. The additional processor and memory usage during interpretation means that a Java application performs more slowly than a native application. The JIT compiler helps improve the performance of Java programs by compiling bytecodes into native machine code at run time.

The JIT compiler is enabled by default, and is activated when a Java method is called. The JIT

compiler compiles the bytecodes of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it. Theoretically, if compilation did not require processor time and memory usage, compiling every method could allow the speed of the Java program to approach that of a native application.
JIT compilation does require processor time and memory usage. When the JVM first starts up, thousands of methods are called. Compiling all of these methods can significantly affect startup time, even if the program eventually achieves very good peak performance

b) Memory is one of the most important resources on a computing system, and its management is primary in every environment. In a bid to use memory efficiently and effectively a number of techniques have been developed to properly manage it. One of these memory management techniques is known as Memory Segmentation (MS). **Memory Segmentation** is defined as a system of segmenting processes and loading them into different non-contiguous addressed spaces in memory. They are referenced using memory addresses. The processes are first 'segmented' (or split) most commonly into three segments, one to house the data, another to house the code and a third to house the stack. This is seen in Figure 1.

Figure 1: A Segmented Program



Programmers may use different variables to achieve this in their program development. The data segment represents all the variables which will be used in running the program. The code segment is the actual execution of the process, while the stack segment monitors the progress and status of the different elements of the program. Now, depending on how complex a program is or its level of sophistication the program may be comprised of many more segments.
Once the process of segmentation occurs, the entire process can be loaded into different areas in memory instead of one contiguous space. This allows for the loading of smaller segments of the process into memory, allowing the physical memory to be used more efficiently. This loading is done by a placement algorithm with processes provided the exact memory space they require as in dynamic partitioning. This technique allows for better memory management, reducing the occurrences of fragmentation. Programmers armed with this technique segment their programs according to the corresponding program logic. This makes segmentation more realistic to the programmer

c) Distributed Deadlock Avoidance
As in centralized system, distributed deadlock avoidance handles deadlock prior to occurrence. Additionally, in distributed systems, transaction location and transaction control issues needs to be addressed. Due to the distributed nature of the transaction, the following conflicts may occur –

- Conflict between two transactions in the same site.
- Conflict between two transactions in different sites.

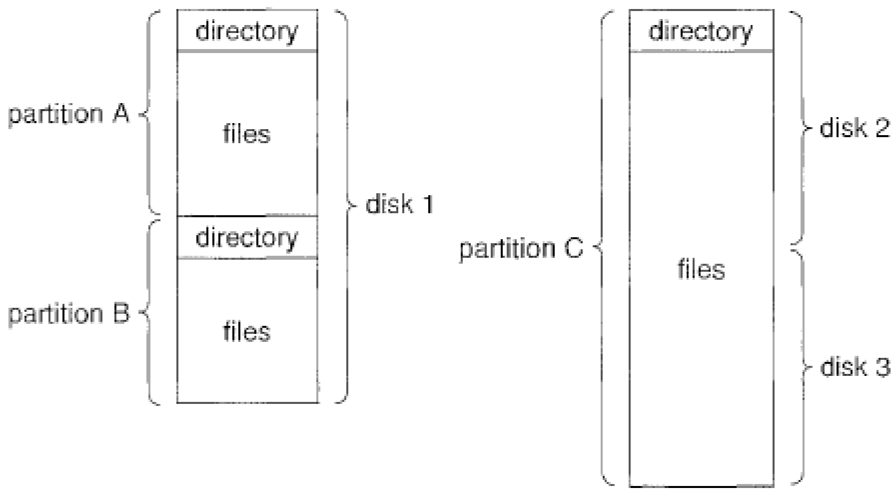
In case of conflict, one of the transactions may be aborted or allowed to wait as per distributed wait-die or distributed wound-wait algorithms.

		Let us assume that there are two transactions, T1 and T2. T1 arrives at Site P and tries to lock a data item which is already locked by T2 at that site. Hence, there is a conflict at Site P. The algorithms are as follows −
		• **Distributed Wound-Die**
		  ○ If T1 is older than T2, T1 is allowed to wait. T1 can resume execution after Site P receives a message that T2 has either committed or aborted successfully at all sites.
		  ○ If T1 is younger than T2, T1 is aborted. The concurrency control at Site P sends a message to all sites where T1 has visited to abort T1. The controlling site notifies the user when T1 has been successfully aborted in all the sites.
		• **Distributed Wait-Wait**
		  ○ If T1 is older than T2, T2 needs to be aborted. If T2 is active at Site P, Site P aborts and rolls back T2 and then broadcasts this message to other relevant sites. If T2 has left Site P but is active at Site Q, Site P broadcasts that T2 has been aborted; Site L then aborts and rolls back T2 and sends this message to all sites.
		If T1 is younger than T1, T1 is allowed to wait. T1 can resume execution after Site P receives a message that T2 has completed processing.

	d)	Schedulers Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types − • Long-Term Scheduler • Short-Term Scheduler • Medium-Term Scheduler Long Term Scheduler It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler. Short Term Scheduler It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers. Medium Term Scheduler Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes. A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

e) File System Organization

A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control. For example, a disk can be into quarters, and each quarter can hold a file system. Storage devices can also be collected together into RAID sets that provide protection from the failure of a single disk. Sometimes, disks are subdivided and also collected into RAID sets. Partitioning is useful for limiting the sizes of individual file systems, putting multiple file-system types on the same device, or leaving part of the
device available for other uses, such as swap space or unformatted (rz;c:.v) disk space. Partitions are also known as or (in the IBM world) A file system can be created on each of these parts of the disk. Any entity containing a file system is generally known as a The volume may be a subset of a device, a whole device, or multiple devices linked together into a RAID set. Each volume can be thought of as a virtual disk. Volumes can also store multiple operating systems, allowing a system to boot and run more than one
operating system. Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a or ~ The device directory (more commonly known simply as that records information -such as name, location, size, and type-for all files on that volume. Figure below shows a typical file-system organization.



f) Two Phase Locking Protocol

This protocol ensures serializability is the This protocol requires that each transaction issue lock and unlock requests in two phases:

- Growing phase. A transaction may obtain locks but may not release any locks.
- Shrinking phase. A transaction may release locks but may not obtain any new locks.

Initially a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and no more lock requests can be issued.

The two-phase locking protocol ensures conflict serializability It does not, however, ensure freedom from deadlock. In addition, it is possible that, for a given set of transactions, there are conflict-serializable schedules that cannot be obtained by use of the two-phase locking protocol. To improve performance over two-phase locking, we need either to have additional information about the transactions or to impose some structure or ordering on the set of data.