END TERM EXAMINATION [MAY-JUNE 2017]
SIXTH SEMESTER [B.TECH]
OPERATING SYSTEMS [ETCS–304]

Time : 3 Hrs. **M.M. : 75**

Note: *Attempt any five questions including Q. No. 1 which is compulsory. Select one question from each unit.*

Q.1. Answer the following short answer questions:

Q.1. (a) What do you mean by context switching ? **(2.5)**

Ans. Context Switching Definition:

A **context switch** (also sometimes referred to as a process **switch** or a task**switch**) is the **switching** of the CPU (central processing unit) from one process or thread to another. A process (also sometimes referred to as a task) is an executing (i.e., running) instance of a program.

Q.1.(b) Explain the concept of pre-emption.Where it is needed ? **(2.5)**

Ans. Pre-emption refers to the temporary interruption and suspension of a task, without asking for its cooperation, with the intention to resume that task later. This act is called a context switch and is typically performed by the pre-emptive scheduler, a component in the operating system authorized to pre-empt, or interrupt, and later resume tasks running in the system.

Needed:

Making a scheduler preemptible has the advantage of better system responsiveness and scalability, but comes with the disadvantage of race conditions (where the executing process accesses the same resource before another preempted process finished using it)

Q.1. (c) Discuss various types of resources ? **(2.5)**

Ans. Operating system resources are the physical or virtual components of limited availability within a computer system. Every device connected to a computer system is a resource. Every internal system component is a resource. Virtual system resources include files, network connections and memory areas.

CPU time, memory (random access memory as well as virtual memory), secondary storage like hard disks, network throughput, battery power, external devices are all resources of a computer which an operating system manages.

Q.1.(d) Describe the role of TLB in address translation. **(2.5)**

Ans. A translation lookaside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

When a virtual memory address is referenced by a program, the search starts in the CPU. First, instruction caches are checked. If the required memory is not in these very fast caches, the system has to look up the memory's physical address. At this point, TLB is checked for a quick reference to the location in physical memory.

When an address is searched in the TLB and not found, the physical memory must be searched with a memory page crawl operation. As virtual memory addresses are translated, values referenced are added to TLB. When a value can be retrieved from TLB, speed is enhanced because the memory address is stored in the TLB on processor. Most processors include TLBs to increase the speed of virtual memory operations through the inherent latency-reducing proximity as well as the high-running frequencies of current CPU's.

TLBs also add the support required for multi-user computers to keep memory separate, by having a user and a supervisor mode as well as using permissions on read and write bits to enable sharing.

TLBs can suffer performance issues from multitasking and code errors. This performance degradation is called a cache thrash. Cache thrash is caused by an ongoing computer activity that fails to progress due to excessive use of resources or conflicts in the caching system.

Q.1. (e) What is the utility of cache memory in the system ? **(2.5)**

Ans. Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chipthat has a separate bus interconnect with the CPU.

The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program.

Q.1.(f) What is scheduling criteria for CPU scheduler ? **(2.5)**

Ans : There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:

- **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- **Throughput** - Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.
- **Turnaround time** - Time required for a particular process to complete, from submission time to completion. (Wall clock time.)
- **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
- (**Load average** - The average number of processes sitting in the ready queue waiting their turn to get into the CPU. Reported in 1-minute, 5-minute, and 15-minute averages by "uptime" and "who".)
- **Response time** - The time taken in an interactive program from the issuance of a command to the ***commence*** of a response to that command.

Q.1.(g) What do you mean by deadlock? Is it possible to have a deadlock in system involving only single process? **(2.5)**

Ans. A **deadlock** is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. The earliest computer operating systems ran only one program at a time.

Is it possible to have a deadlock in system involving only single process:

Deadlock with one process is not possible. Here is the explanation.

A deadlock situation can arise if the following four conditions hold simultaneously in a system.

- Mutual Exclusion.
- Hold and Wait.
- No Preemption.
- Circular-wait.

It is not possible to have circular wait with only one process, thus failing a necessary condition for Circular wait. There is no second process to form a circle with the first one. So it is not possible to have a deadlock involving only one process.

Q.1.(h) Discuss various methods for maintaining free spaces on disk. (2.5)

Ans. The main idea behind allocation is effective utilization of file space and fast access of the files. There are three types of allocation:

1. contiguous allocation
2. linked allocation
3. indexed allocation

In addition to storing the actual file data on the disk drive, the file system also stores metadata about the files: the name of each file, when it was last edited, exactly where it is on the disk, and what parts of the disk are "free". Free areas are not currently in use by the file data or the metadata, and so available for storing new files. (The places where this metadata is stored are often called "inodes", "chunks", "file allocation tables", etc.)

To keep track of the free space, the file system maintains a free-space list which tracks all the disk blocks which are free. To create a file, the required space is reserved for the file and the corresponding space is removed from the free list linked to each other.

Q.1.(i) Why is page size always a power of 2 ? **(2.5)**

Ans. It is most efficient to break the address into X **page** bits and Y offset bits, rather than perform arithmetic on the address to calculate the **page** number and offset. Because each bit position represents a **power of 2**, splitting an address between bits results in a **page size** that is a **power of 2**.

Q.1. (j) What is Inode Table ? Describe its contents. **(2.5)**

Ans. An **inode** is an entry in **inode table**, containing information (the metadata) about a regular file and directory. An **inode** is a data structure on a traditional Unix-style file system such as ext3 or ext4. **Inode** number also called as index number.

Contents :

Inodes store information about files and directories (folders), such as file ownership, access mode (read, write, execute permissions), and file type. On many types of file system implementations, the maximum number of inodes is fixed at file system creation, limiting the maximum number of files the file system can hold. A typical allocation heuristic for inodes in a file system is one percent of total size.

The inode number indexes a table of inodes in a known location on the device. From the inode number, the kernel's file system driver can access the inode contents, including the location of the file - thus allowing access to the file.

A file's inode number can be found using the ls -i command. The ls -i command prints the i-node number in the first column of the report.

UNIT-I

Q.2.(a) Describe working of paging memory management scheme. Compare paging with segmentation. **(6.5)**

Ans. Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- *Logical Address or Virtual Address (represented in bits):* An address generated by the CPU
- *Logical Address Space or Virtual Address Space(represented in words or bytes):* The set of all logical addresses generated by a program
- *Physical Address (represented in bits):* An address actually available on memory unit
- *Physical Address Space (represented in words or bytes):* The set of all physical addresses corresponding to the logical addresses

Example:

- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})

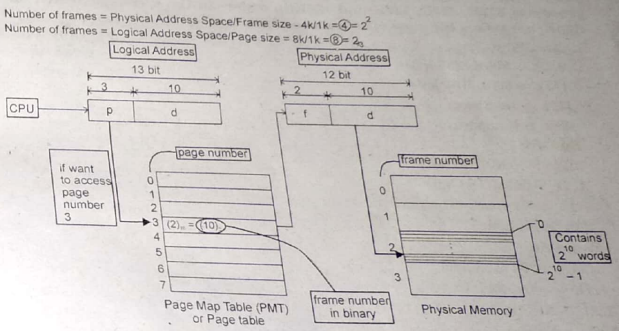
- If Logical Address Space = 128 M words = $2^7 \times 2^{20}$ words, then Logical Address = $\log_2 2^{27} = 27$ bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^4 \times 2^{20}$ words, then Physical Address = $\log_2 2^{24} = 24$ bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)



Address generated by CPU is divided into

- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Basis for Comparison	Paging	Segmentation
Basic Fragmentation	A page is of fixed block size. Paging may lead to internal fragmentation.	A segment is of variable size. Segmentation may lead to external fragmentation.
Address	The user specified address is divided by CPU into a page number and offset.	The user specifies each address by two quantities a segment number and the offset (Segment limit).
Size	The hardware decides the page size.	The segment size is specified by the user.
Table	Paging involves a page table that contains base address of each page.	Segmentation involves the segment table that contains segment number and offset (segment length).

Q.2. (b) Given memory partitions of 100 K, 500 K and 600 K (in order). Where will the algorithm, namely,best fit, first fit and worst fit place the processes 212K, 417K, 112K and 426K(in order) in the memory? Which one algorithm makes the most efficient use of memory? (4)

Ans. First-fit:

212k -> 500K (288 left)
417k -> 600k (183 left)
112k -> No Space
426k -> No Space

Best-fit:

212k -> 500K (288 left)
417k -> 600k (183 left)
112k -> No Space
426k -> No Space

Worst-fit:

212k -> 600k (388 left)
417k -> 500k (83 left)
112k -> No Space
426k -> No Space

The best fit and First Fit algorithms uses memory most efficiently.

Q.2.(c) What is the cause of thrashing? What can the system do to eliminate the problem? (2)

Ans. Meaning of Thrashing

If the process does not have number of frames it needs to support pages in active use, it will quickly page-fault. The high paging activity is called thrashing.

In other words we can say that when page fault ratio decreases below level, it is called thrashing.

Causes of Thrashing

It result in severe performance problems.

(1) If CPU utilization is too low then we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is

used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.

(2) CPU utilization is plotted against the degree of multiprogramming.

(3) As the degree of multiprogramming increases, CPU utilization also increases.

(4) If the degree of multiprogramming is increased further, thrashing sets in and CPU utilization drops sharply.

(5) So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

Eliminate the problem: Thrashing is caused by under allocation of the minimum numberof pages required by a process, forcing it to continuously page fault.The system can detect thrashing by evaluating the level of CPU utilizationas compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

Q.3.(a) 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 . How many page faults would occur for following replacement algorithm (Assume three frames): (5)

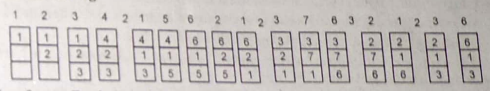
(i) FIFO

(ii) LRU

(iii) Optimal

Ans. (i) FIFO: A FIFO replacement algorithm assosicates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page in chosen.

Reference string

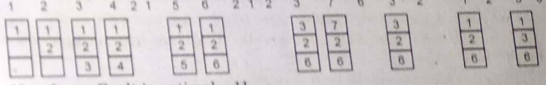


No. of page Fault in FIFO = 16

(ii) LRU: If we use the recent past as an approximation of the near future, then we will replace the page that has not been used for the longest period of time. This approach is the Least Recently used.

No. of page fault in LRU = 15

(iii) Optimal: Replace the page that has not been used for a longest period of time.



No. of page Fault in optimal = 11

Q.3. (b) What do you mean by a page fault? What actions are taken by the operating system when a page fault occurs? (3.5)

Ans: A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

Though the term "page fault" sounds like an error, page faults are common and are part of the normal way computers handle virtual memory. In programming terms, a page fault generates an exception, which notifies the operating system that it must retrieve the memory blocks or "pages" from virtual memory in order for the program to continue. Once the data is moved into physical memory, the program continues as normal. This process takes place in the background and usually goes unnoticed by the user.

Most page faults are handled without any problems. However, an invalid page fault may cause a program to hang or crash. This type of page fault may occur when a program tries to access a memory address that does not exist. Some programs can handle these types of errors by finding a new memory address or relocating the data. However, if the program cannot handle the invalid page fault, it will get passed to the operating system, which may terminate the process. This can cause the program to unexpectedly quit.

While page faults are common when working with virtual memory, each page fault requires transferring data from secondary memory to primary memory. This process may only take a few milliseconds, but that can still be several thousand times slower than accessing data directly from memory. Therefore, installing more system memory can increase your computer's performance, since it will need to access virtual memory less often.

Steps taken by operating system to handle a page fault

Here is the sequence of steps which occurs in handling a page fault:

(1) Hardware traps to kernel mode.

(2) Program counter is saved in stack.

(3) Current instruction state is saved in CPU registers.

(4) An assembly code program is run to save general registers and other information.

(5) Operating system founds that page fault has occurred.

(6) It finds a virtual address which caused the page fault by using a hardware register or by using the software.

(7) Virtual address is checked to see if it valid. If it is not valid then process is killed.

(8) It is also checked to see if a protection fault has occurred.

(9) If virtual address is valid and if protection fault has not occurred, then, operating system checks if a frame is free.

(10) If there is no frame free, then page replacement algorithm is run.

(11) If selected page frame is dirty, page is scheduled for transfer to disk.

(12) Context switch happens and the process which caused the fault is suspended and another process is run.

(13) Frame is marked as busy.

(14) When page frame is clean, then the page is loaded from the disk.

(15) When the page has been loaded, page frame is set in normal state.

(16) Instruction that caused the fault is set to initial state program counter is reset to point this instruction.

(17) Process which caused the fault is scheduled.

(18) Operating system runs assembly language program to reload registers and other state information.

(19) Control is shifted back to user space for execution.

Q.3. (c) Differentiate the following : - (4)

(i) Networked and Distributed operating system.

(ii) Multiprogramming and Multi-Processing operating system.

Ans: (i) Network and Distributed Operating systems have a common hardware base, but the difference lies in software.

S.No.	Network Operating System	Distributed Operating System
1	A network operating system is made up of software and associated protocols that allow a set of computer network to be used together.	A distributed operating system is an ordinary centralized operating system but runs on multiple independent CPUs.
2	Environment users are aware of multiplicity of machines.	Environment users are not aware of multiplicity of machines.
3	Control over file placement is done manually by the user.	It can be done automatically by the system itself.
4	Performance is badly affected if certain part of the hardware starts malfunctioning.	It is more reliable or fault tolerant i.e distributed operating system performs even if certain part of the hardware starts malfunctioning.
5	Remote resources are accessed by either logging into the desired remote machine or transferring data from the remote machine to user's own machines.	Users access remote resources in the same manner as they access local resources.

(*ii*) Following are the differences between multiprocessing and multiprogramming.

Ans.

S.No.	Multiprocessing	Multiprogramming
1	Multiprocessing refers to processing of multiple processes at same time by multiple CPUs.	Multiprogramming keeps several programs in main memory at the same time and execute them concurrently utilizing single CPU.
2	It utilizes multiple CPUs.	It utilizes single CPU.
3	It permits parallel processing.	Context switching takes place.
4	Less time taken to process the jobs.	More Time taken to process the jobs.
5	It facilitates much efficient utilization of devices of the computer system.	Less efficient than multiprocessing.
6	Usually more expensive.	Such systems are less expensive.

UNIT-II

Q.4. (a) Consider the following set of processes, with the length of the CPU-burst time give in milliseconds: **(6.5)**

Process	Burst Time	Priority
P1	8	3
P2	1	1
P3	2	3
P4	4	2
P5	5	4

The process are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0. Draw the Grant chart and find the average turn around time and average

waiting time of each process for the following scheduling algorithmes. (i) FCS (ii) Priority (Non-preemptive) (iii) RR (quantum of 2m)

Ans.

Process	Burst Time	Priority
P1	8	3
P2	1	1
P3	2	3
P4	4	2
P5	5	4

(i) FCFS :

TAT = CT – AT
WT = TAT – BT

Grantt Chart :

P1	P2	P3	P4	P5
0	8	9	11	15	20

Process	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P1	0	8	8	8	0
P2	0	1	9	9	8
P3	0	2	11	11	9
P4	0	4	15	15	11
P5	0	5	20	20	15

Average Turn Around Time $= \frac{8+9+11+15+20}{5} = \frac{63}{5}$

$= 12.60$ milliseconds.

Average Waiting Time $= \frac{0+8+9+11+15}{5} = \frac{43}{5}$

$= 8.60$ milliseconds.

(ii) Priority (Non-Preemptive): Lowest number yet higher priority.

Gantt Chart:

P2	P4	P1	P3	P5
0	1	5	13	15	20

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT = CT–AT	Waiting time (WT) WT = TAT – BT
P1	0	8	13	13	5
P2	0	1	1	1	0
P3	0	2	15	15	13
P4	0	4	5	5	1
P5	0	5	20	20	15

Average TAT $= \frac{13+1+15+5+20}{5} = \frac{54}{5} = 10.80$ milliseconds

Average WT $= \frac{5+0+13+1+15}{5} = \frac{34}{5} = 6.80$ milliseconds

(iii) RR (quantum of 2 ms):

Gantt chart:

P1	P2	P3	P4	P5	P1	P4	P5	P1	P5	P1
0	2	3	5	7	9	11	13	15	17	18	20

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT = CT–AT	Waiting time (WT) WT = TAT – BT
P1	0	8	20	20	12
P2	0	1	3	3	2
P3	0	2	5	5	3
P4	0	4	13	13	9
P5	0	5	18	18	13

Average TAT $= \frac{20+3+5+13+18}{5} = \frac{59}{5} = 11.80$ milliseconds

Average WT $= \frac{12+2+3+9+13}{5} = \frac{39}{5} = 7.80$ milliseconds

Q.4. (b) Describe Process State Diagram. **(3)**

Ans. To start off with the details, a program when needs to be executed goes through a process. This process has several state changes in the entire operation until termination of the program. Upon successful termination, the program would get useful results to the user. This entire process progression goes through state changes which are mention below in steps:

1. The process enters a state called **NEW STATE**.
2. The process then enters the **READY STATE**.
3. The process then goes to an **ACTIVE STATE/RUNNING STATE** (Execution of the program starts here).

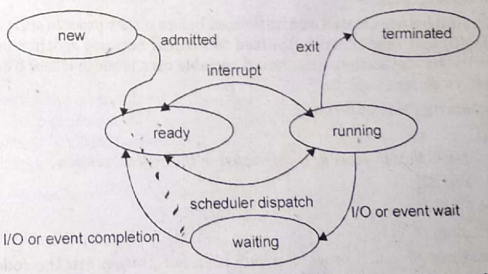


Fig. Process State Diagram with Entire Operation

4.The process ends with the **HALT STATE/TERMINATED STATE** (after all the program BURST's are over. The program might be terminated forcibly or else is terminated normally).

Q.4. (c) What is the role of Medium Term Scheduler? **(3)**

Ans. The use of medium term scheduler is to improve multiprogramming by allowing multiple processes to reside in main memory by swapping out processes that are waiting (need I/O) or low priority processes and swapping in other processes that were in ready queue. So you can see that we requied medium term scheduler when we have limited memory. This swappin g in and out operation does not take place when we are running a single small program and have large memory.

Similary if we are running multiple programs and we have very large memory(larger than the size of all processes plus addition space for other requirements) then medium term scheduler is not needed. Modern operating systems use paging so instead of swapping processes they swap pages in and out of memory.It is same as a system with very large memory(infinite) would not suffer from page faults.

Q.5. (a) What do you mean by Critical Section? What are various methods to handle critical section problem? Write a solution for Dining Philosophers problem using Semaphores. **(8.5)**

Ans. Critical Section

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.

Solutions to the Critical Section Problem

Assumption

- assume that a variable (memory location) can only have one value; never "between values"
- if processes A and B write a value to the same memory location at the "same time," either the value from A or the value from B will be written rather than some scrambling of bits

Peterson's Algorithm

- a simple algorithm that can be run by two processes to ensure mutual exclusion for one resource (say one variable or data structure)
- does not require any special hardware
- it uses busy waiting (a spinlock)

Peterson's Algorithm

Shared variables are created and initialized before either process starts. The shared variables flag[0] and flag[1] are initialized to FALSE because neither process is yet interested in the critical section. The shared variable turn is set to either 0 or 1 randomly (or it can always be set to say 0).

var flag: **array** [0..1] **of** boolean;
turn: 0..1;
%flag[k] means that process[k] is interested in the critical section
flag[0] := FALSE;
flag[1] := FALSE;
turn := random(0..1)

After initialization, each process, which is called process *i* in the code (the other process is process *j*), runs the following code:

repeat
flag[i] := TRUE;
turn := j;
while (flag[j] **and** turn=j) **do** no-op;
CRITICAL SECTION
flag[i] := FALSE;
REMAINDER SECTION
until FALSE;

Information common to both processes:
turn = 0
flag[0] = FALSE
flag[1] = FALSE
EXAMPLE 1

Process 0	**Process 1**
i = 0, j = 1 flag[0] := TRUE turn := 1 check (flag[1] = TRUE and turn = 1) - Condition is false because flag[1] = FALSE- - Since condition is false, no waiting in while loop - Enter the critical section - Process 0 happens to lose the processor	i = 1, j = 0
	flag[1] := TRUE turn := 0 check (flag[0] = TRUE and turn = 0)- - Since condition is true, it keeps busy waiting until it loses the processor
- Process 0 resumes and continues until it finishes in the critical section - Leave critical section flag[0] : = FALSE - Start executing the remainder (anything else a process does besides using the critical section) - Process 0 happens to lose the processor	
	check (flag[0] = TRUE and turn = 0) - This condition fails because flag[0] = FALSE - No more busy waiting - Enter the critical section

EXAMPLE 2

Process 0	Process 1
i=0, j=1 flag[0] = TRUE turn = 1 - Lose processor here	i=1, j=0
	flag[1] := TRUE turn := 0 check (flag[0] = TRUE and turn = 0) Condition is true so Process 1 busy waits until it loses the processor
check (flag[1] = TRUE and turn = 1) - This condition is false because turn = 0 - No waiting in loop - Enters critical section	

EXAMPLE 3

Process 0	Process 1
i=0, j=1 flag[0] = TRUE - Lose processor here	i=1, j=0
	flag[1] = TRUE turn = 0 check (flag[0] = TRUE and turn = 0) - Condition is true so, Process 1 busy waits until it loses the processor
turn:=1 check (flag[1] = TRUE and turn = 1 - Condition is true so process 0 busy waits until it loses the processor	
	check (flag[0] = TRUE and turn = 0) - The condition is false so, Process 1 enters the critical section

Swap

- another type of hardware assistance for process synchronization
- a special hardware instruction that does a complete swap (ordinarily three operations) **atomically**
- i.e., the complete swap is executed or no part of it is
- given pass-by-reference parameters A and B, it swaps their values

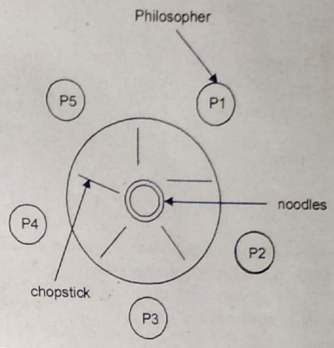
Swap Solution to the Critical Section Problem

- uses two variables called lock and key
- intuition: if lock is false, then a process can enter the critical section, and otherwise it can't
- initially, lock is false

repeat

```
var = true
while (var == true) swap(lock, var);
critical section
lock = false;
remainder section
until false
```


The Dining Philosopher Problem – The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.



Semaphore Solution to Dining Philosopher –

Each philosopher is represented by the following pseudocode:

```
process P[i]
 while true do
  { THINK;
    PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
    EAT;
    PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
  }
```

There are three states of philosopher : **THINKING, HUNGRY and EATING**. Here there are two semaphores : Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or putdown at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

Code –

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
```

```
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];
void test(int phnum)
{
if (state[phnum] == HUNGRY
&& state[LEFT] != EATING
&& state[RIGHT] != EATING) {
// state that eating
state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
printf("Philosopher %d is Eating\n", phnum + 1);

// sem_post(&S[phnum]) has no effect
// during takefork
// used to wake up hungry philosophers
// during putfork
sem_post(&S[phnum]);
    }
}
// take up chopsticks
void take_fork(int phnum)
{
sem_wait(&mutex);
// state that hungry
state[phnum] = HUNGRY;
printf("Philosopher %d is Hungry\n", phnum + 1);
// eat if neighbours are not eating test(phnum);
sem_post(&mutex);
// if unable to eat wait to be signalled
sem_wait(&S[phnum]);
sleep(1);
}
// put down chopsticks
void put_fork(int phnum)
{
    sem_wait(&mutex);
//state that thinking
state[phnum] = THINKING;
printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);
printf("Philosopher %d is thinking\n", phnum + 1);
```



```
test(LEFT);
test(RIGHT);
sem_post(&mutex);
}
void* philospher(void* num)
{
            while (1) {
             int* i = num;
            sleep(1);
            take_fork(*i);
            sleep(0);
            put_fork(*i);
           }
}

 int main()
 {
 int i;
pthread_t thread_id[N];
// initialize the semaphores
 sem_init(&mutex, 0, 1);
for (i = 0; i < N; i++)
    sem_init(&S[i], 0, 0);
for (i = 0; i < N; i++) {
// create philosopher processes
pthread_create(&thread_id[i],NULL, philospher, &phil[i]);
printf("Philosopher %d is thinking\n", i + 1);
  }
for (i = 0; i < N; i++)
    pthread_join(thread_id[i], NULL);
}
```

Q.5. (b) Discuss Dekker's Algorithm. **(4)**

Ans. Dekker's algorithm was the first provably-correct solution to the critical section problem. It requires both an array of boolean values and an integer variable:

```
var flag: array [0..1] of boolean;
 turn: 0..1;
 repeat
    flag[i] := true;
    while flag[j] do
       if turn = j then
       begin
          flag[i] := false;
    while turn = j do no-op;
    flag[i] := true;
 end;
 critical section
 turn := j;
```

flag[i] := false;
remainder section
until false;

UNIT-III

Q.6.(a) Consider the following snapshot of a system.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Answer the following questions using Banker's algorithm. **(6.5)**

(i) Is the system in a safe state?

(ii) If a request from process P1 arrives for (3,3,0) can the request be granted immediately?

Ans.

Process	Allocation A B C	Max A B C	Available A B C	Need A B C
P0	0 1 0	7 5 3	3 3 2	7 4 3
P1	2 0 0	3 2 2		1 2 2
P2	3 0 2	9 0 2		6 0 0
P3	2 1 1	2 2 2		0 1 1
P4	0 0 2	4 3 3		4 3 1

(i) The content of the matrix need is defined to be MAX – Allocation and is above.

We claim that the system is currently in safe state. Indeed, the sequence <P1, P3, P4, P2, P0> satisfies the safety criteria.

(ii) You should be able to see, however, that, when the system is in this state, a request for (3,3,0) by P4 cannot be granted, since the resources are not available. A request for (0,2,0) by P0 cannot be granted, even though the resources are available, since the resulting state is unsafe.

Q.6.(b) What is the difference between deadlock avoidance and prevention? **(3)**

Ans : Prevention:

- The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.
- Deadlock prevention is often impossible to implement.
- The system doesnot require additional apriori information regarding the overall potential use of each resource for each process.
- In order for the system to prevent the deadlock condition it does not need to know all the details of all resources in existence, available and requested.
- Deadlock prevention techniques include non-blocking synchronization algorithms, serializing tokens, Dijkstras algorithm etc.

- Resource allocation strategy for deadlock prevention is conservative, it under commits the resources.
- All resources are requested at once.
- In some cases preempts more than often necessary.

Avoidance:

- The goal for deadlock avoidance is to the system must not enter an unsafe state.
- Deadlock avoidance is often impossible to implement.
- The system requires additional apriori information regarding the overall potential use of each resource for each process.
- In order for the system to be able to figure out whether the next state will be safe or unsafe, it must know in advance at any time the number and type of all resources in existence, available, and requested.
- Deadlock avoidance techniques include Banker's algorithm, Wait/Die, Wound/Wait etc.
- Resource allocation strategy for deadlock avoidance selects midway between that of detection and prevention.
- Needs to be manipulated until atleast one safe path is found.
- There is no preemption.

Q.6. (c) How can the no-preemption and circular wait conditions be prevented? **(3)**

Ans. No Preemption

- Preemption of process resource allocations can prevent this condition of deadlocks, when it is possible.

o One approach is that if a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, (preempted), forcing this process to re-acquire the old resources along with the new resources in a single request, similar to the previous discussion.

o Another approach is that when a resource is requested and not available, then the system looks to see what other processes currently have those resources *and* are themselves blocked waiting for some other resource. If such a process is found, then some of their resources may get preempted and added to the list of resources for which the process is waiting.

o Either of these approaches may be applicable for resources whose states are easily saved and restored, such as registers and memory, but are generally not applicable to other devices such as printers and tape drives.

Circular Wait

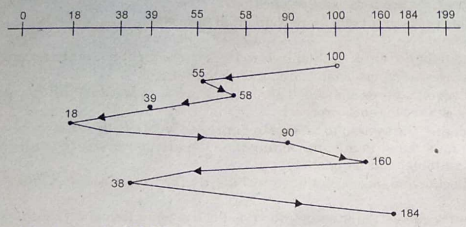
- One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order.
- In other words, in order to request resource Rj, a process must first release all Ri such that i >= j.
- One big challenge in this scheme is determining the relative ordering of the different resources

Q.7. (a) Suppose that a disk drive has 200 tracks numbered 0 to 199. The drive is currently serving a request at track number 100. The requested tracks, in order received by the disk scheduler are 55,58,39,18,90,160,38,184. What is the total distance (in tracks) that disk arm moves to satisfy all the pending requests, for each of the following disk scheduling algorithms? **(6.5)**

(i) FCFS
(ii) SSTF
(iii) SCAN
(iv) C-SCAN

(v) LOOK

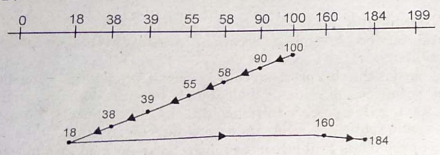
Ans: (i) ECFS:



The total distance in FCFS:

(100 – 55) + (58 – 55) + (58 – 39) + (39 – 18) + (90 – 18) +
(160 – 90) + (160 – 38) + (184 – 38) = 498 cylinders.

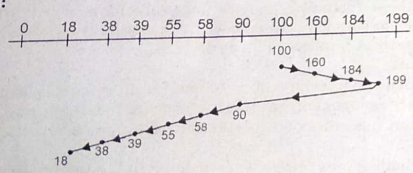
(ii) SSTF:



The total distance in SSTF:

(100 – 90) + (90 – 58) + (58 – 55) + (55 – 39) + (39 – 38)
+ (38 – 18) + (160 – 18) + (184 – 160) = 248 cylinders

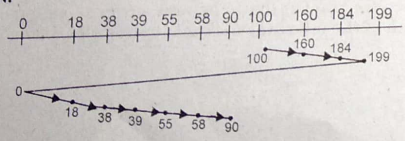
(iii) SCAN:



Total distance in SCAN:

(160 – 100) + (184 – 160) + (199 – 184) + (199 – 90) + (90 – 58) +
(58 – 55) + (55 – 39) + (39 – 38) + (38 – 18) = 280 cylinders

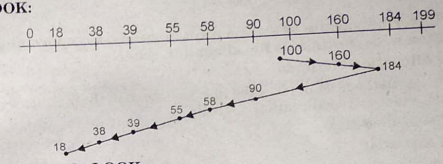
(iv) C-SCAN:



Total distance in C-SCAN:

(160 – 100) + (184 – 160) + (199 – 184) + (199 – 0) + (18 – 0) + (38 – 18)
+ (39 – 38) + (55 – 39) + (58 – 55) + (90 – 58) = 388 cylinders

(v) LOOK:



Total distance in LOOK

(160 – 100) + (184 – 160) + (184 – 90) + (90 – 58) + (58 – 55) +
(55 – 39) + (39 – 38) + (38 – 18) = 250 cylinders

Q.7.(b) What are various parameters for evaluating Disk performance? Discuss. **(4)**

Ans. Higher performance in hard disk drives comes from devices which have better performance characteristics. These devices include those with rotating media, hereby called *rotating drives*, i.e., hard disk drives (HDD), floppy disk drives (FDD), optical discs (DVD-RW / CD-RW), and it also covers devices without moving parts like solid-state drives(SSD). For SSDs, most of the attributes related to the movement of mechanical components are not applicable, but the device is actually affected by some other electrically based element that still causes a measurable delay, when isolating and measuring that attribute.These performance characteristics can be grouped into two categories: access time and data transfer time (or rate).

Access Time:

The *access time* or *response time* of a rotating drive is a measure of the time it takes before the drive can actually transfer data. The factors that control this time on a rotating drive are mostly related to the mechanical nature of the rotating disks and moving heads. It is composed of a few independently measurable elements that are added together to get a single value when evaluating the performance of a storage device.

The key components that are typically added together to obtain the access time are:

- Seek time
- Rotational latency
- Command processing time
- Settle time

Data Transfer Rate:

The *data transfer rate* of a drive (also called *throughput*) covers both the internal rate (moving data between the disk surface and the controller on the drive) and the external rate (moving data between the controller on the drive and the host system). The measurable data transfer rate will be the lower (slower) of the two rates.

Q.7.(c) Why rotational latency is usually not considered in disk scheduling? **(2)**

Ans. Most disks do not export their rotational position information to the host. Even if they did, the time for this information to reach the scheduler would be subject to imprecision and the time consumed by the scheduler is variable, so the rotational

position information would become incorrect. Further, the disk requests are usually given in terms of logical block numbers, and the mapping between logical blocks and physical locations is very complex.

UNIT–IV

Q.8. (a) Describe various file allocation methods. Compare and contrast index allocation with contiguous file allocation scheme. (4.5)

Ans. File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

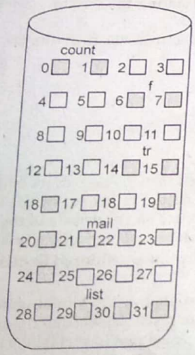
1. Contiguous Allocation

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.



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

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:

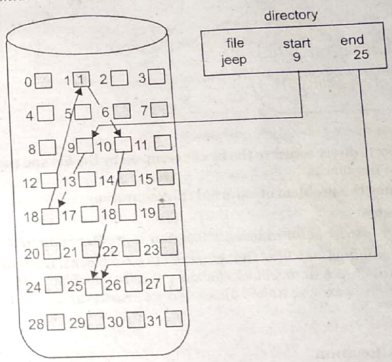
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

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.

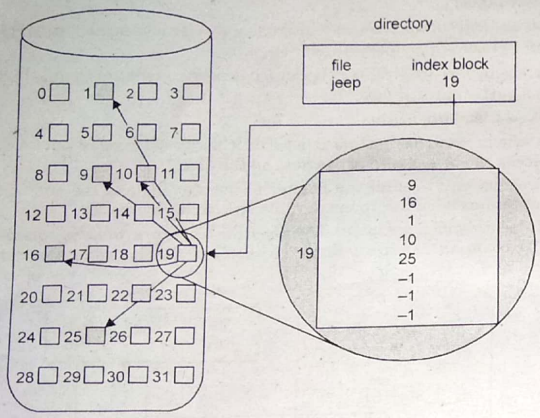
Disadvantages:

- 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.

Comaprison:

Index Block Allocation

1. File block addresses are stored in an array which is stored in a disk block
2. Directory has a pointer to index block
3. Both random and sequential access are easy
4. File size limitation depends on array size for example if we have (256) pointers in the index block then the maximum file size will be 256KB assuming 1KB data blocks
5. Space utilization is good
6. Files can grow or shrink easily

Using the same assumptions as before here is how to map from logical to physical address:

1. block = index[pos/1024]
2. offset = pos%1024

Linked Index Block Allocation

1.Similar to linked file blocks but using index blocks instead of data blocks

2. Good for large files

3.Both random and sequential access are easy

4.Random access is slow for large files

Q.8. (b)Why directory structure is required? Discuss various types of directory structures along with respective merits and demerits. (4)

Ans. In computing, a **directory structure** is the way an operating system's file system and its files are displayed to the user. Files are typically displayed in a hierarchical tree structure.

There are many types of directory structure in Operating System. They are as follows

(1) Single Level Directory
(2) Two Level Directory
(3) Tree Structured Directory
(4) Acyclic Graph Directory
(5) General Graph Directory

(1)Single Level Directory

In Single Level Directory all files are in the same directory.

Limitations of Single Level Directory

(a) Since all files are in the same directory, they must have unique name.
(b) If two user call their data free test, then the unique name rule is violated.
(c) Files are limited in length.
(d) Even a single user may find it difficult to remember the names of all files as the number of file increases.
(e) Keeping track of so many file is daunting task.

(2) Two Level Directory

(i) Each user has Its own User File Directory (UFD).
(ii) When the user job start or user log in, the system Master File Directory (MFD) is searched. MFD is indexed by user name or Account Number.
(iii) When user refers to a particular file, only his own UFD is searched.

Thus different users may have files with same name. To have a particular file uniquely, in a two level directory, we must give both the user name and file name.

A two level directory can be a tree or an inverted tree of height 2

The root of a tree is Master File Directory (MFD).

Its direct descendents are User File Directory (UFD). The descendents of UFD's are file themselves.

The files are the leaves of the tree.

Limitations of Two Level Directory

The structure effectively isolates one user from another.

(3) Tree Structured Directory

A directory (or Sub directory) contains a set of files or sub directories. All directories has the same internal format.

(i) One bit in each directory entry defines the entry.
(ii) Special calls are used to create and delete directories.

(iii) Each process has a current directory. Current directory should contain most of the files that are of current interest to the process.

(iv) When a reference is made to a file, the current directory is searched.

(v) The user can change his current directory whenever he desires.

(vi) If a file is not needed in the current directory then the user usually must either specify a path name or change the current directory.

Paths can be of two types :-

(a) *Absolute Path*

Begins at root and follows a path down to the specified file.

(b) *Relative Path*

Defines a path from current directory.

(vii) Deletions if directory is empty, its entry in the directory that contains it can simply deleted. If it is not empty : One of the Two approaches can be taken :-

(a)User must delete all the files in the directory.

(b)If any sub directories exist, same procedure must be applied.

The **UNIX rm command** is used.

MS dos will not delete a directory unless it is empty.

(4) Acyclic Graph Directory

Acyclic Graph is the graph with no cycles. It allows directories to share sub directories and files. With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the another.

Implememtation of Shared files and directories

(i) *To create a link*

- A link is effectively a pointer to another file or sub directory.
- Duplicate all the information about them in both sharing directories.

(ii) *Deleting a link*

- Deletion of the link will not effect the original file, only link is removed.
- To preserve the file until all references to it are deleted.

Q.8. (c) How data integrity is maintained? Explain. (4)

Ans. To guarantee data integrity, organizations need to establish strong quality management practices that will help protect and maintain data during collection, processing and storage.

Data Cleaning and Maintenance

Research by The Data Warehouse Institute (TDWI) reported that data quality issues can cost US businesses more than $600 billion annually. Yet decision-makers do not take action with their bad data until it manifests itself into high-impact costly problems. An essential first step in producing information that translates into business performance and profitability is data cleaning.

A data cleaning approach should satisfy several requirements. First of all, it should detect, eliminate or correct all errors and inconsistencies. It should also be a continuous process that supports system health in order to maintain data integrity. As a proactive solution, the Data Integrity Gateway (DIG) tool integrates with an institution's information system and centralizes cleanup projects in a single repository. By automating processes, delegating tasks, and monitoring data cleanup, DIG helps maintain data quality throughout its life-cycle.

Data Entry Training & Accountability

Data integrity starts at the source – the user. Manual data entry can result in errors that compromise analytical results meant to guide business decisions. That's why it is vital that staff members with system access are properly trained on data entry and upload protocols. There are several steps to consider when training:

- Training should be an active, evolving process in response to operational needs.
- An easy-to-understand document with procedures should be readily available for reference.
- System administrators should assign correct level of access to users based on their training and role.
- Auditing processes should be put into place so that individuals can be held accountable for any inaccurate data entered into the system.

Data Validation Rules

Even with a proper training plan in place, there is always room for human-error when a company includes manual data entry in their operations. By using data validation rules, administrators can ensure data integrity by controlling and restricting the values that users can enter into their system. By protecting information from accidental alteration, validation rules provide additional security and data quality assurance - a natural requirement for accurate analytics.

Q.9. (a) An operating system only supports a single directory but allows that directory to have arbitrarily many files with arbitrarily long file names. Can an approximate hierarchical file system be simulated? How? (4.5)

Ans. One way to simulate that is to append to each file name the name of directory that contains it.

For example: UsrStudentsZivaPrivateFileX

"Hierarchical File System" (HFS) is the file system used for organizing files on a Macintosh hard disk. When a hard disk is formatted for a Macintosh computer, the hierarchical file system is used to create a directory that can expand as new files and folders are added to the disk. Since HFS is a Macintosh format, Windows computers cannot recognize HFS-formatted drives. Windows hard drives are typically formatted using WIN32 or NTFS file systems.

Since HFS was not originally designed to handle large hard disks, such as the 100GB+ hard disks that are common today, Apple introduced a updated file system called HFS+, or HFS Extended, with the release of Mac OS 8.1. HFS+ allows for smaller clusters or block sizes, which reduces the minimum size each file must take up. This means disk space can be used much more efficiently on large hard disks. Mac OS X uses the HFS+ format by default and also supports journaling, which makes it easier to recover data in case of a hard drive crash.

Q.9. (b) Some systems support many types of structures for file's data, while others simply support a stream of bytes. What are the advantages and disadvantages? (4)

Ans. An advantage of having the system support different file structures is that the support comes from the system; individual applications are not required to provide the support. In addition, if the system provides the support for different file structures, it can implement the support presumably more efficiently than an application. The disadvantage of having the system provide support for defined file types is that it increases the size of the system. In addition, applications that may require different file types other than what is provided by the system may not be able to run on such systems. An alternative strategy is for the operating system to define no support for file structures and instead treat all files as a series of bytes. This is the approach taken by UNIX systems. The advantage of this approach is that it simplifies the operating system support for file systems, as the system no longer has to provide the structure for different file types. Furthermore, it allows applications to define file structures, thereby

alleviating the situation where a system may not provide a file definition required for a specific application.

Q.9.(c) Describe file access control mechanism. (4)

Ans : Access control is a security technique that can be used to regulate who or what can view or use resources in a computing environment.

There are two main types of access control: physical and logical. Physical access control limits access to campuses, buildings, rooms and physical IT assets. Logical access limits connections to computer networks, system files and data.

The four main categories of access control are:

- Mandatory access control
- Discretionary access control
- Role-based access control
- Rule-based access control

Access control systems perform authorization identification, authentication, access approval, and accountability of entities through login credentials including passwords, personal identification numbers (PINs), biometric scans, and physical or electronic keys.