OPERATING SYSTEMS

B. Tech. II – I

UNIT - 1

Operating System Overview and Process Management

1 . Define System Call ? What is user mode vs kernel mode ? What are the types of system calls , list few of the system calls under each category ?

Ans .

System Call :

A system call is a request from computer program/software to an operating system's kernel.

System calls provide an interface to the services made available by an operating system.

User Mode	Kernel Mode
User mode is the mode in which the applications are running	Kernel mode is the privileged mode to which the computer enters when accessing hardware resources.
The system is in user mode when the operating system is running a user application such as handling a text editor.	The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode.
The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs.	These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated.

Types Of System Calls :

Here are the types of system calls −

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

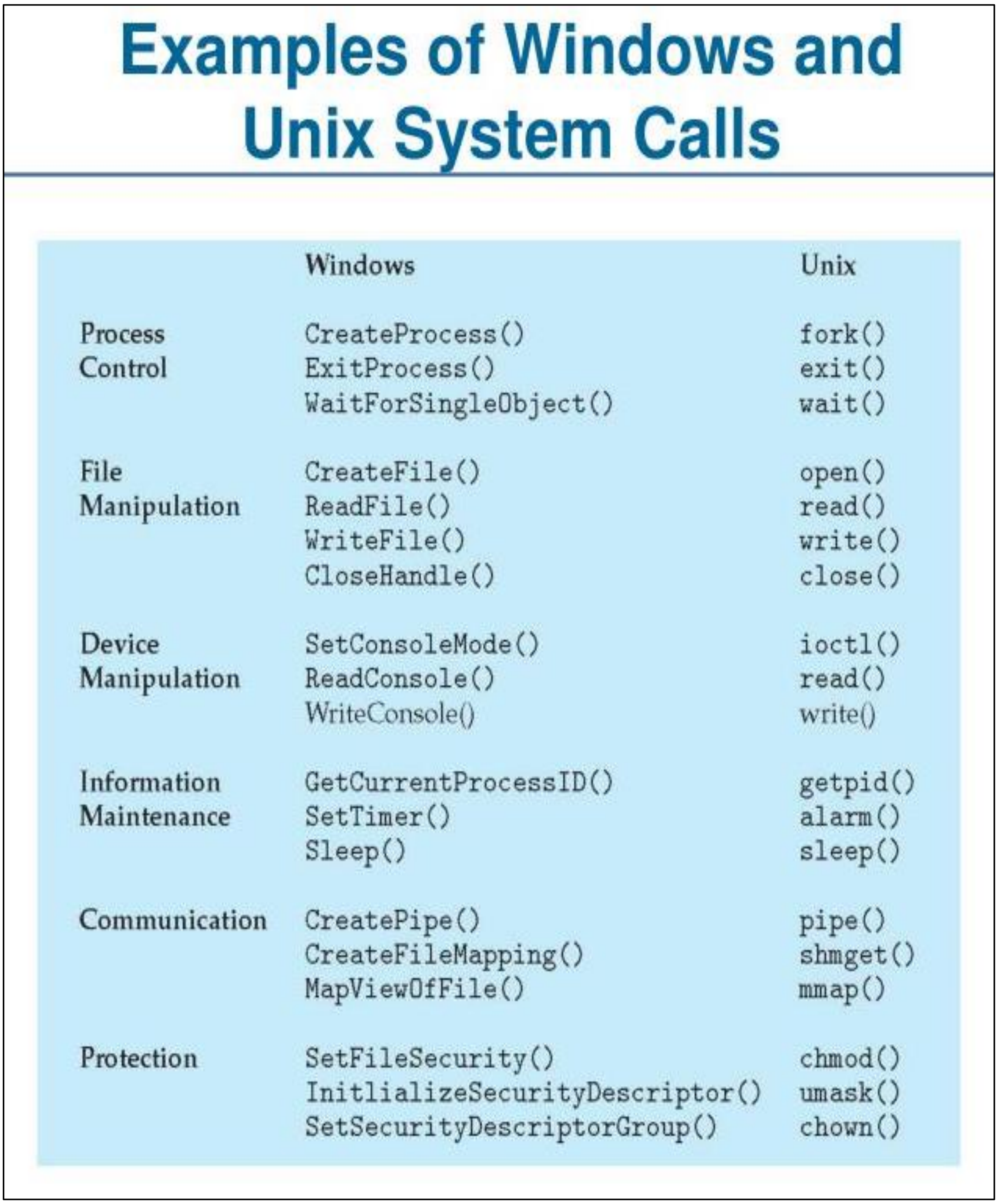
Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for inter process communication. They also deal with creating and deleting a communication connection.

System Calls Under Each Category :



Examples of Windows and Unix System Calls

	Windows	Unix

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Manipulation	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device Manipulation	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

2 .Define OS ? List few OS services ? What are the types of OS /Evolution of OS ?

Ans.

Operating System :

An operating system is a System Software that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

OS Services :

Services provided by OS are –

1).User Interface :

OS provides either

a) Command line interface
b) Batch based interface
c) Graphical user interface

2).Program Execution :

The OS must have the capability to load a program into memory and execute that program. Further more the program must be able to end its execution either normally or abnormally.

3).File system manipulation :

Programs needs to be read and then write them into files and directories. File Handling process of an OS allows users to create and delete file by specific names along with extensions.

4).Input and Output operations :

A program which is currently executing may require I/O which may ivolve file or other I/O devices. For efficiency and protection users cant control IO devices. So, the OS provides a mean to do I/ operations with any file.

5).Communication and Service :

Process need to be swap or share the information among them. Process executing on same computer system or an another computer system can communicate with each other using OS support . This communication can be done using shared memory:

6).Resource Allocation :

When multiple jobs are running concurrent resources must nedd to be allocated to each of them. Resources can be main memory storage , file storage , IO devices , etc. CPU scheduling routines are used to establish an efficient way of utilization of Cpu.

7).Error Detection :

Errors may occur with in cpu , memory hardware , IO devices and in the user program. For each type of error , OS takes adequate action for ensuring correct and consistent computing.

8).Accounting :

This service of the OS keeps track of which users are using how much and what kind of compute resources have been used for accounting.

Types / Evolution of OS :

1.Mainframe Systems :

Reduce setup time by batching similar jobs Automatic job sequencing – automatically transfers control from one job to another. First rudimentary operating system monitor :

⇐ initial control in monitor

⇐control transfers to job

⇐ when job completes control transfers pack to monitor

2. Batch Processing Operating System:

⇐ This type of OS accepts more than one jobs and these jobs are grouped together according to their similar requirements. This is done by computer operator. Whenever the computer becomes available, the batched jobs are sent for execution and gradually the output is sent back to the user. ⇐ It allowed only one program at a time.

⇐ This OS is responsible for scheduling the jobs according to priority and the resource required.

3. Multiprogramming Operating System:

⇐ This type of OS is used to execute more than one jobs simultaneously by a single processor. it increases CPU utilization by organizing jobs , so that the CPU always has one job to execute.

⇐ The concept of multiprogramming is described as follows:

1 - In a multiprogramming system, the operating system simply switches to another job and executes. When that job needs to wait, the CPU is switched to another job, and so on.

2 - When the first job finishes waiting and it gets the CPU back.

3 - As long as at least one job needs to execute, the CPU is never idle. Multiprogramming operating systems use the mechanism of job scheduling and CPU scheduling.

4. Time-Sharing/multitasking Operating Systems :

Time sharing (or multitasking) OS is a logical extension of multiprogramming. It provides extra facilities such as:

⇐ Faster switching between multiple jobs to make processing faster.

⇐ Allows multiple users to share computer system simultaneously.

⇐ The users can interact with each job while it is running. These systems use a concept of virtual memory for effective utilization of memory space.

5. Multiprocessor Operating Systems :

Multiprocessor operating systems are also known as parallel OS or tightly coupled OS. Such operating systems have more than one processor in close communication that sharing the computer bus, the clock and sometimes memory and peripheral devices. It executes multiple jobs at same time and makes the processing faster. Multiprocessor systems have three main advantages:

⇐ Increased throughput

⇐ Economy of scale

⇐ Increased reliability

The multiprocessor operating systems are classified into two categories:

1. Symmetric multiprocessing system

2. Asymmetric multiprocessing system

⇐ In **symmetric multiprocessing system**, each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

⇐ In **asymmetric multiprocessing system**, a processor is called master processor that controls other processors called slave processor. Thus, it establishes master-slave relationship. The master processor schedules the jobs and manages the memory for entire system.

5. Distributed Operating Systems :

⇐ In distributed system, the different machines are connected in a network and each machine has its own processor and own local memory.

⇐ In this system, the operating systems on all the machines work together to manage the collective network resource.

⇐ It can be classified into two categories:

   1. Client-Server systems

   2. Peer-to-Peer systems

7. Real-Time Operating Systems (RTOS) :

⇐ A real-time operating system (RTOS) is a multitasking operating system intended for applications with fixed deadlines.

⇐ The real time operating system can be classified into two categories:

1. hard real time system

2. soft real time system.

⇐ A **hard real-time system** guarantees that critical tasks be completed on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made of it.

⇐ A **soft real-time system** is a less restrictive type of real-time system. Here, a critical real-time task gets priority over other tasks and retains that priority until it completes. Due to less restriction, they are risky to use for industrial control.

3 What is Program vs Process ? Explain PCB in detail ? Discuss Process State Transition Diagram ?

Ans.

Program vs Process :

1) Both are same beast with different name or when this beast is sleeping (not executing) it is called program and when it is executing becomes process.

2) Program is a static object whereas a process is a dynamic object.

3) A program resides in secondary storage whereas a process resides in main memory.

4) The span time of a program is unlimited but the span time of a process is limited.

5) A process is an 'active' entity whereas a program is a 'passive' entity.

6) A program is an algorithm expressed in programming language whereas a process is expressed in assembly language or machine language.

PCB :

Process Control Block (PCB) Information associated with each process.

• Process state

• Program counter

• CPU registers

• CPU scheduling information

• Memory-management information

• Accounting information

• I/O status information

Process state: The state may be new, ready, running, waiting, halted, and so on.

Program counter: The counter indicates the address of the next instruction to be executed for this process.

CPU registers: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

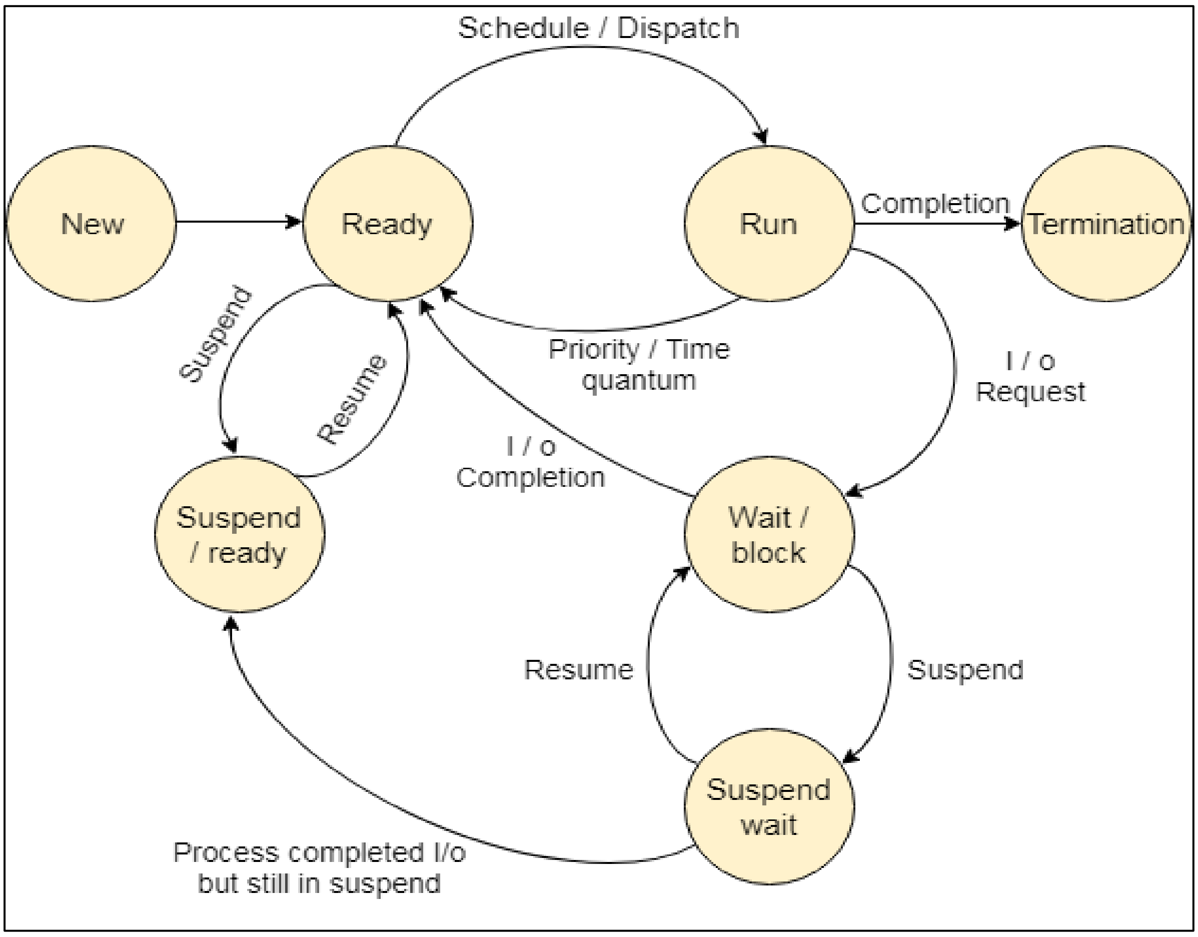
CPU-scheduling information: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

Accounting information: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

Status information: The information includes the list of I/O devices allocated to this process, a list of open files, and so on. The PCB simply serves as the repository for any information that may vary from process to process.

Process State Transition Diagram :



Process State

As a process executes, it changes state

⇐ **New State:** The process is being created.

⇐ **Running State:** A process is said to be running if it has the CPU, that is, process actually using the CPU at that particular instant. ⇐ **Blocked (or waiting) State:** A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that a process is unable to run until some external event happens.

⇐ **Ready State:** A process is said to be ready if it needs a CPU to execute. A ready state process is runnable but temporarily stopped running to let another process run.

⇐ **Terminated state:** The process has finished execution.

4 Que. What is Thread ? Difference between Process & Thread ? Explain Different Kernel Level Threads ?

Ans .

Thread :

A thread, sometimes called a lightweight process (LWP), is a basic unit of CPU utilization.

it comprises a thread ID, a program counter, a register set, and a stack.

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

If the process has multiple threads of control, it can do more than one task at a time.

Process vs Thread :

Process	**Thread**
1.Process cannot share the same memory area(address space)	1.Threads can share memory and files.
2.It takes more time to create a process.	2.It takes less time to create a thread.
3.It takes more time to complete the execution and terminate.	3.Less time to terminate.
4.Execution is very slow.	4.Execution is very fast.
5.It takes more time to switch between two processes.	5.It takes less time to switch between two threads.
6.System calls are required to communicate each other.	6.System calls are not required.
7.It requires more resources to execute.	7.Requires fewer resources.
8.Implementing the communication between processes is bit more difficult.	8.Communication between two threads are very easy to implement because threads share the memory

Kernel-Level Threads :

In this method, the kernel knows about and manages the threads. Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system. Operating Systems kernel provides system call to create and manage threads.

Advantages:

• Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads.

• Kernel-level threads are especially good for applications that frequently block.

Disadvantages:

• The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.

since kernel must manage and schedule threads as well as processes. It require a full thread control block (TCB) for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity.

Multi-level Threads:

Multithreading Models Many systems provide support for both user and kernel threads, resulting in different multithreading models. We look at three common types of threading implementation.

Many-to-One Model:

The many-to-one model maps many user-level threads to one kernel thread. Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call. Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.

One-to-one Model:

The one-to-one model maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors. The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of threads supported by the system.

Many-to-Many Model :

The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine (an application may be allocated more kernel threads on a multiprocessor than on a uniprocessor). Whereas the many-to-one model allows the developer to create as many user threads as she wishes, true concurrency is not gained because the kernel can schedule only one thread at a time. The one-to-one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application.

5. Define Scheduling ? List different parameters of Scheduling ? Discuss different Scheduling algorithms.

a) FCFS b) SJF c)Priority (Preemptive , non-Preemptive)

d) Round Robin

Ans.

Scheduling :

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Different parameters of Scheduling :

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithms :

a) FCFS

First-Come, First-Served Scheduling :

- By far the simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm.
- With this scheme, the process that requests the CPU first is allocated the CPU first.
- The implementation of the FCFS policy is easily managed with a FIFO queue.
    - When a process enters the ready queue, its PCB is linked onto the tail of the queue.
    - When the CPU is free, it is allocated to the process at the head of the queue.
    - The running process is then removed from the queue.
- The code for FCFS scheduling is simple to write and understand.
- The average waiting time under the FCFS policy, however, is often quite long.

b) SJF

Shortest Job First :

- A different approach to CPU scheduling is the shortest-job-first (SJF) scheduling algorithm.
- This algorithm associates with each process the length of the latter's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.

- If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie.
- Note that a more appropriate term would be the shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of a process, rather than its total length.

c) Priority :

- The SJF algorithm is a special case of the general priority-scheduling algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.
- We discuss scheduling in terms of high priority and low priority. Priorities are generally some fixed range of numbers, such as 0 to 7,However, there is no general agreement on whether 0 is the highest or lowest priority.

Priority scheduling can be either pre-emptive or non-pre-emptive.

❖ When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.

  - A **Pre-emptive priority**-scheduling algorithm will pre-empt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.
  - A **Non-Pre-emptive priority**-scheduling algorithm will simply put the new process at the head of the ready queue.

❖ A major problem with priority-scheduling algorithms is indefinite blocking (or starvation).

d) Round Robin :

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but pre-emption is added to switch between processes.
- A small unit of time, called a time quantum (or time slice), is defined. A time quantum is generally from 10 to 100 milliseconds.
- The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

- One of two things will then happen.
    - The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue.
    - Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system.
- A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

UNIT -2

Synchronization, Deadlocks

1.Define race condition, critical section and process synchronization ? What are the necessary conditions for process Synchronization Explain them? Discuss two process synchronization solutions.

(i)Test and set (ii) Peterson solution

A:

Race condition:

A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition.**

Critical section:

The portion of the program where the shared data variables or shared resources or shared data will be placed is called a **critical section.**

Consider a system consisting of n processes {Po,P1, ..., Pn-1). Each process has a segment of code, called a **critical section**, in which the process may be changing common variables, updating a table, writing a file, and so on.

The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

Process synchronization:

Process Synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources. (critical section).

It is specially needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or data at the same time.

The Necessary conditions for process synchronization are:

1)Mutual exclusion

2)Progress

3)Bounded waiting

4)Architectural Neutral

Mutual Exclusion:

Mutual Exclusion means that if one process is executing inside critical section then the other process must not enter in the critical section.

Mutual Exclusion is a Primary condition.

For example: If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

Progress:

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.

Progress is also a Primary condition.

Bounded Waiting:

There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

The process must not be endlessly waiting for getting into the critical section.

Bounded waiting is a secondary condition.

Architectural Neutral:

Our Synchronization mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Architectural Neutral is a secondary condition.

Process synchronization solutions:

1)Test and set:

It is a hardware solution to the synchronization problem.

There is a shared lock variable which can take either of the two values 0 or 1.

Before entering into the critical section, a process inquires about the lock.

If it is locked, it keeps on waiting till it becomes free.

If it is not locked, it takes the lock and executes the critical section.

Test and set solution:

TestAndSet():

```
do {
boolean TestAndSet(boolean *target){
boolean rv=*target;
*target=TRUE;
return rv;
}
```

Process P1:

```
do {
while(TestAndSet(&lock));
//do nothing
//critical section
lock=FALSE;
//remainder section
}while(TRUE);
```

Process P2:

```
do {
while(TestAndSet(&lock));
//do nothing
//critical section
lock=FALSE;
//remainder section
}while(TRUE);
```

ADV:

Satisfies mutual exclusion.

DISADV:

Does not satisfy bounded-waiting.

2)Peterson solution:

Peterson’s solution is a classic software based solution to the critical section problem.

It provides a good algorithmic description of solving the critical section problem and illustrates some of the complexities involved in designing software that addresses the requirements of mutual exclusion, progress, and bounded waiting requirements.

It may not work correctly on modern computer architectures.

Peterson’s solution is restricted to two process that alternate execution between their critical sections and remainder sections.

Peterson solution:

Peterson’s solution requires two data items to be shared between the two processes:

1)**turn(int):**indicates whose turn it is to enter its critical section.

2)**flag(Boolean):**used to indicate if a process is ready to enter its critical section.

Consider two processes Pi and Pj.

```
Pi:                                   Pj:
do{                                     do{
flag[i]=true;                         flag[j]=true;
turn=j;                               turn=i;
while(flag[j] && turn==[j]);     while(flag[i] && turn==[i]);
//critical section                    //critical section
Flag[i]=false;                          Flag[j]=false;
//remainder section                 //remainder section
}while(TRUE);                         }while(TRUE);
```

2.Explain the classical synchronization problems with semaphore solutions?

A:

The classical synchronization problems are:

1)Bounded buffer problem.

2)Reader writer problem.

3)Dining philosopher.

1)The Bounded-Buffer Problem

The bounded-buffer problem is commonly used to illustrate the power of synchronization primitives. We present here a general structure of this scheme, without committing ourselves to any particular implementation.

We assume that the pool consists of n buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers, respectively. The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.

The code for the **producer process** is

```
do{
produce an item in nextp
...
wait (empty) ;
wait (mutex);
…
add nextp to buffer
. . .
signal(mutex);
signal (full) ;
}while (1);
```

The code for the **consumer** process is

```
do{
```

```
wait (full) ;
wait (mutex) ;
…
remove an item from buffer to nextc
...
signal (mutex) ;
signal (empty) ;
...
consume the item in nextc
...
} while (1);
```

The Readers- Writers Problem

A data object (such as a file or record) is to be shared among several concurrent processes. Some of these processes may want only to read the content of the shared object, whereas others may want to update (that is, to read and write) the shared object. We distinguish between these two types of processes by referring to those processes that are interested in only reading as readers, and to the rest as writers. Obviously, if two readers access the shared data object simultaneously, no adverse effects will result. However, if a writer and some other process (either a reader or a writer) access the shared object simultaneously, chaos may ensue.

To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared object. This synchronization problem is referred to as the **readers-writers problem.**

R-R (No problem)

W-R(Problem)

W-W(Problem)

R-W(Problem)

Solutions to the readers writers problem using semaphores:

We will make use of two semaphores and an integer variable:

1.mutex (Semaphore (initialized to 1) which is used to ensure mutual exclusion.

2.wrt (semaphore (initialized to 1) common to both reader and writer.)

3.readcount (integer variable (initialized to 0) that keeps track of how many process are currently reading the object.)

The code for a **writer** process is

```
do{
wait (wrt) ;
. . .
writing is performed
...
signal(wrt);
}while(1);
```

The code for a **reader** process is

```
do{
wait (mutex) ;
readcount++;
if (readcount == 1)
wait (wrt) ;
signal (mutex) ;
. . .
reading is performed
...
wait (mutex) ;
readcount--;
if (readcount == 0)
signal(wrt1;
signal (mutex) ;
}while(1);
```

The Dining-Philosophers Problem

Consider five philosophers who spend their lives thinking and eating. The philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors). A philosopher may pick up only one chopstick at a time. Obviously, she cannot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again.

The structure of philosopher i

```
do {

wait (chopstick[i]) ;

wait (chopstick[(i+1) % 5] ) ;

...

eat

. . .

signal (chopstick [i] ;

signal(chopstick[(i+1) % 5] ) ;

. . .

think

...} while (1);
```

The dining-philosophers problem is considered a classic synchronization problem, neither because of its practical importance nor because computer scientists dislike philosophers, but because it is an example of a large class of concurrency-control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock- and starvation free manner.

One simple solution is to represent each chopstick by a semaphore. A philosopher tries to grab the chopstick by executing a wait operation on that semaphore; she releases her chopsticks by executing the signal operation on the appropriate semaphores. Thus, the shared data are semaphore chopstick [5] ; where all the elements of chopstick are initialized to 1.

Although this solution guarantees that no two neighbors are eating simultaneously, it nevertheless must be rejected because it has the possibility of creating a deadlock. Suppose that all five philosophers become hungry simultaneously, and each grabs her left chopstick. All the elements of chopstick will now be equal to 0. When each philosopher tries to grab her right chopstick, she will be delayed forever. Use an asymmetric solution; that is, an odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick. Finally, any satisfactory solution to the dining-philosophers problem must guard against the possibility that one of the philosophers will starve to death.

3Q) Explain semaphores with their operations in detail?

A:

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread.

It is a mechanism that can be used to provide synchronization of tasks. It is a low-level synchronization mechanism.

Semaphore will always hold a non-negative integer value.

Semaphore can be implemented using test operations and interrupts, which should be executed using file descriptors.

There are two types of semaphores:

1)Counting Semaphore

2)Binary Semaphore

Counting Semaphore:

These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

Binary Semaphore:

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Operations of semaphores:

1)Wait:

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

When a process must wait on a semaphore, it is added to the list of processes.

The wait semaphore operation can now be defined as

```
void wait(semaphore S) {
S--;
if (S< 0) {
add this process to waiting list;
block() ;
}
```

2)Signal:

A signal operation removes one process from the list of waiting processes and awakens that process.

The signal operation increments the value of its argument S.

The signal semaphore operation can now be defined as

```
void signal(semaphore S) {
S++;
if (S <= 0) {
remove a process P from waiting list ;
wakeup (PI) ;
}
```

Advantages of Semaphores

Some of the advantages of semaphores are as follows −

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
- Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores

Some of the disadvantages of semaphores are as follows −

- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

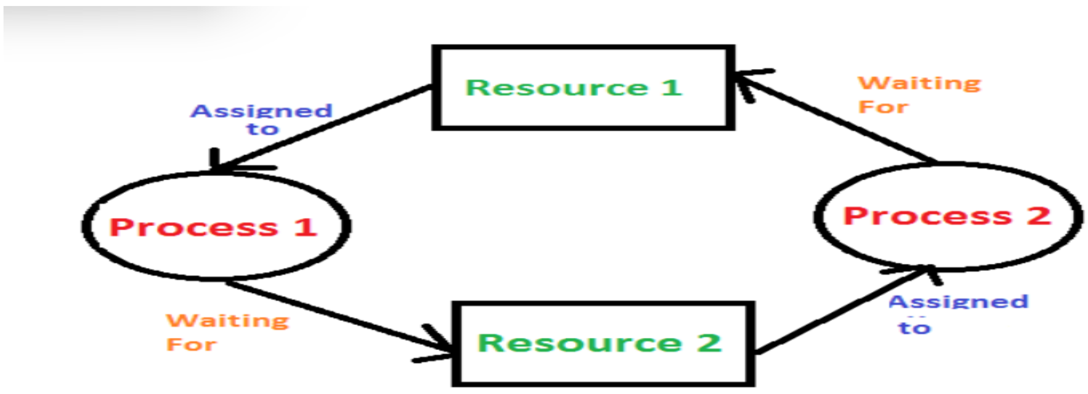
4Q) Define Deadlock? Explain necessary conditions of deadlock? Discuss the deadlock prevention Handling approach in detail?

A:

Deadlock:

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

A set of process is said to be in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. The event can be resource acquisition, resource release etc. The resource can be physical (printers, memory space) or logical (semaphores, files).



Necessary Conditions of deadlock:

1)Mutual Exclusion:

- A resource at a time can only be used by one process.
- If another process is requesting for the same resource, then it must be delayed until that resource is released.

2)Hold and Wait:

- A process is holding a resource and waiting to acquire additional resources that are currently being held by other processes.

3)No Pre-emption:

- Resources cannot be pre-empted

- Resource can be released only by the process currently holding it based on its voluntary decision after completing the task.

4)Circular wait:

- A set of processes { P0,P1,….,Pn-1,Pn } such that the process P0 is waiting for resource held by P1,P1 is waiting for P2 ,and Pn is waiting for P0 to release its resources.
- Every process holds a resource needed by the next process.

All the four above mentioned conditions should occur for a deadlock to occur.

Deadlock Prevention Handling:

The strategy of deadlock prevention is to design the system in such a way that the possibility of deadlock is excluded.

It ensures that the system never enters a deadlock state.

It provides a set of methods to make sure that at least one of the four necessary conditions for a deadlock is never satisfied.

Indirect method prevent the occurrence of one of three necessary condition of deadlock i.e., **mutual exclusion, no pre-emption and hold and wait.**

Direct method prevent the occurrence of **circular wait**.

Mutual exclusion: this condition is needed to be checked for non-sharable resources (e.g. Printer).

Hold and Wait:

This condition can be prevented by requiring that a process requests all its required resources at one time and blocking the process until all of its requests can be granted at a same time simultaneously.

Disadvantage:

1) long waiting time required.
2) in efficient use of allocated resource.
3) A process may not know all the required resources in advance.

No pre-emption : techniques for no pre-emption are:

1)if a process that is holding some resource, requests another resource that can not be immediately allocated to it, the all resource currently being held are released and if necessary, request them again together with the additional resource.
2)If a process requests a resource that is currently held by another process, the OS may pre-empt the second process and require it to release its resources.
Disadvantage:

This works only if both the processes do not have same priority.
Circular wait:
One way to ensure that this condition never hold is to impose a total ordering of all resource types and to require that each process requests resource in an increasing order of enumeration, i.e., if a process has been allocated resources of type R, then it may subsequently request only those resources of types following R in ordering.
Disadvantage of Deadlock Prevention Handling:

1) It can lead to low device utilization.
2) reduced system throughput.

5)Explain Resource allocation graph(RAG) in detail?

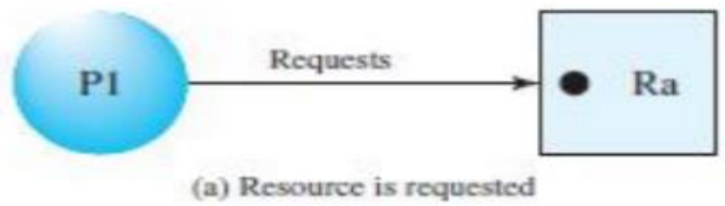
Ans.

Resource-Allocation Graph

Deadlocks can be described in terms of a directed graph called a system resource-allocation graph.

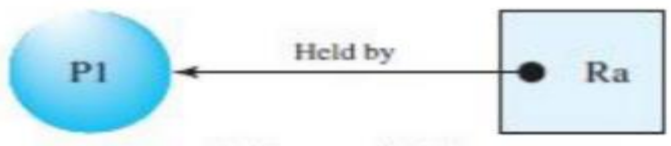
This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes P = {P1, P2, ..., Pn}, the set consisting of all the active processes in the system, and R = {R1, R2, ..., Rm}, the set consisting of all resource types in the system.We represent each process Pi as a circle and each resource type Rj as a square.

A directed edge from process Pi to resource type R j is denoted by Pi→Rj ,it signifies that process Pi requested an instance of resource type Rj and is currently waiting for that resource. A directed edge Pi →Rj is called a request edge.



(a) Resource is requested

A directed edge from resource type Rj to process Pi is denoted by Rj→Pi, it signifies that an instance of resource type Rj has been allocated to process Pi. A directed edge Rj→Pi is called an assignment edge.

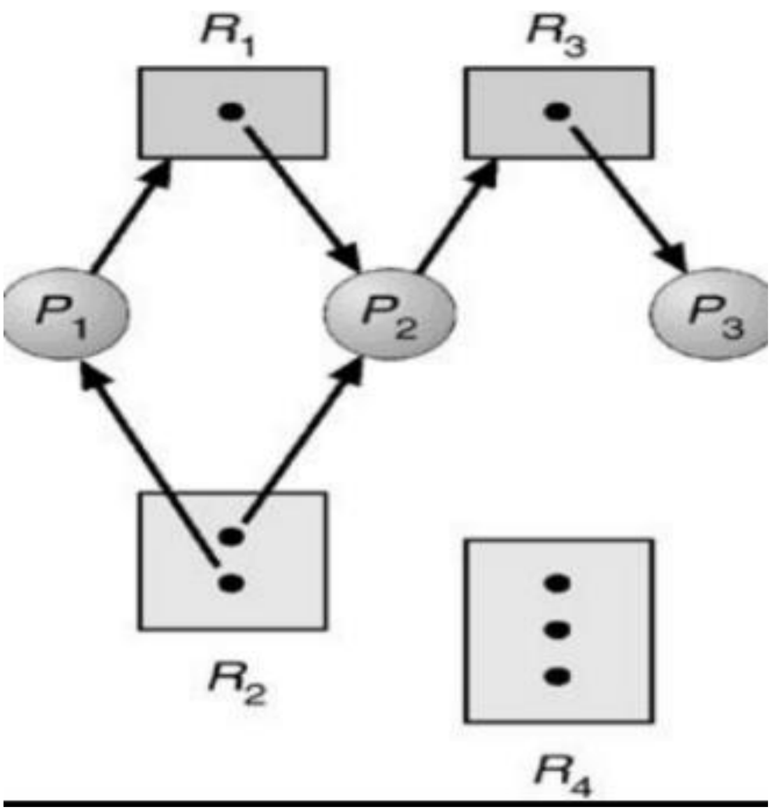


(b) Resource is held

For a resource each instance is represented by a dot within the square.

A resource with one dot is single instance type resource and with more than one dot is multi instance resource.

Example



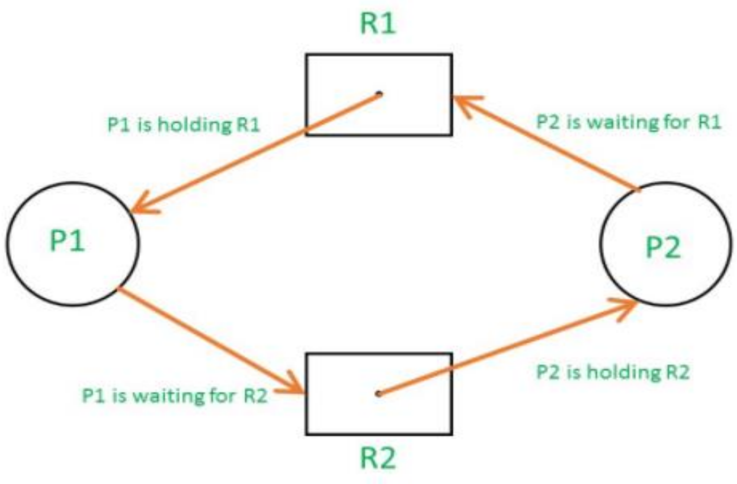
Using Resource Allocation Graph for detection of deadlock

1)In a graph with single instance type resources

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock

A cycle in single instance graph is both a necessary and a sufficient condition for the existence of deadlock.

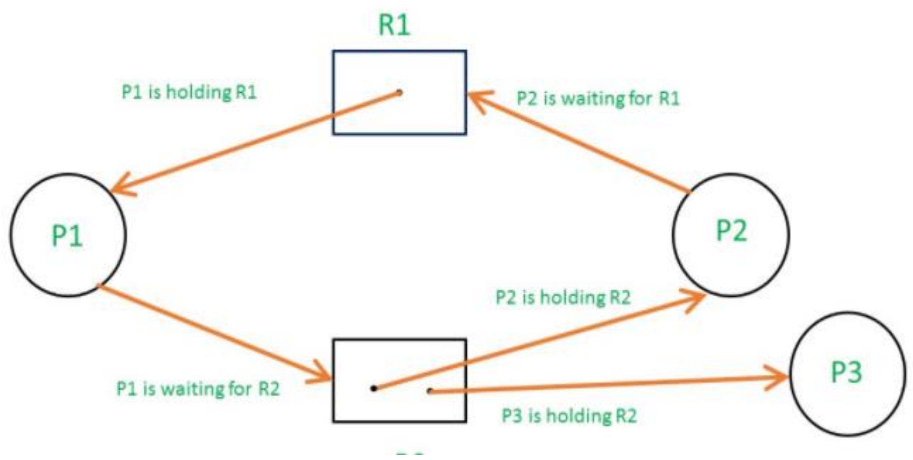
Example:cycle:R1→P1→R2→P2→R1 (below system is in deadlock).



2) In a graph with multi instance type resources

If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

Multi instances without deadlock



we can conclude that if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

6)Explain the Banker’s algorithm (safety algorithm)(or)Deadlock avoidance with an example?

Also explain Request algorithm.

Ans. DEADLOCK AVOIDANCE

This approach to the deadlock problem anticipates deadlock before it actually occurs. If the necessary conditions for a deadlock are in place, it is still possible to avoid deadlock by being careful when resources are allocated.

This approach employs an algorithm to access the possibility that deadlock could occur and acting accordingly.

A system is said to be in a Safe State, if there is a safe execution sequence.

A system is said to be in an Unsafe State, if there is no safe execution sequence.

Banker’s Algorithm

The Banker's algorithm is a resource allocation & deadlock avoidance algorithm used to avoid deadlock and allocate resources safely to each process in the computer system.

Available : Indicates the number of available resources of each type.

Max: defines the maximum demand of each process.

Allocation: defines the number of resources of each type currently allocated to each process.

Need: indicates the remaining resource need of each process.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

n be the number of processes in the system and m be the number of resource types.

Step 1:flag[i]=0 for i=0 to n-1 find need[n][m]

Step 2: Find a process i such that both a.flag[i]==0

b.need i<=available

Step 3:if i exsists then

Flag[i]=1 , available=available+allocated

Goto Step 2

Step 4:if flag[i]==1 for all i then system is safe

Request Algorithm

process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

1. If Request i < Need i, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If Request i < Available, go to step 3. Otherwise, Pi must wait, since the resources are not

available.

3. The requested resources are allocated to process Pi by modifying the state as follows:

Available = Available – Request i;

Allocation i = Allocation i + Request i;

Need i := Need i – Request i;

If the resulting resource-allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the new state is unsafe, then Pi must wait for Request i and the old resource-allocation state is restored.

UNIT -3

Memory Management Strategies, Virtual Memory Management

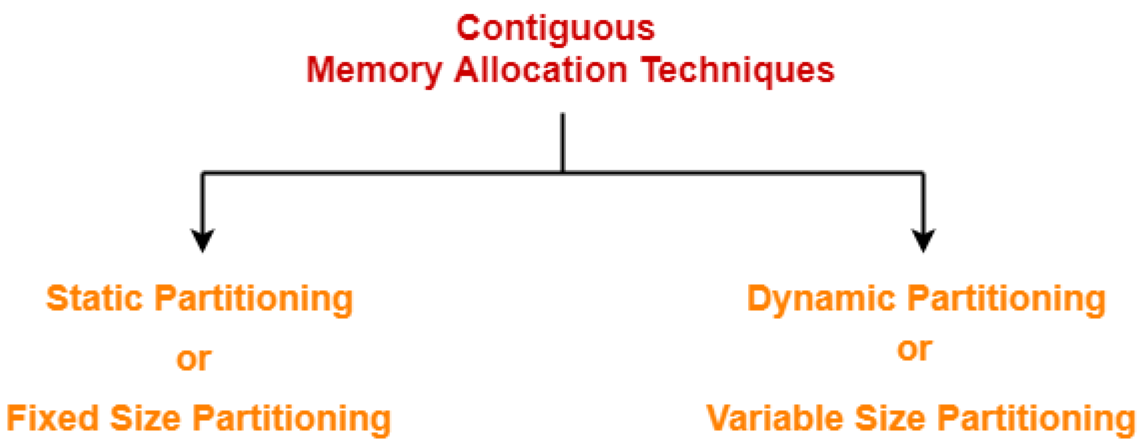
Q1. Explain the contiguous memory allocation techniques with pros and cons in detail. Explain the algorithms to allocate them.

A: **Contiguous Memory Allocation**

- Contiguous memory allocation is a memory allocation technique.
- It allows to store the process only in a contiguous fashion.
- Thus, entire process has to be stored as a single entity at one place inside the memory.

Techniques :

There are two popular techniques used for contiguous memory allocation .They are ,



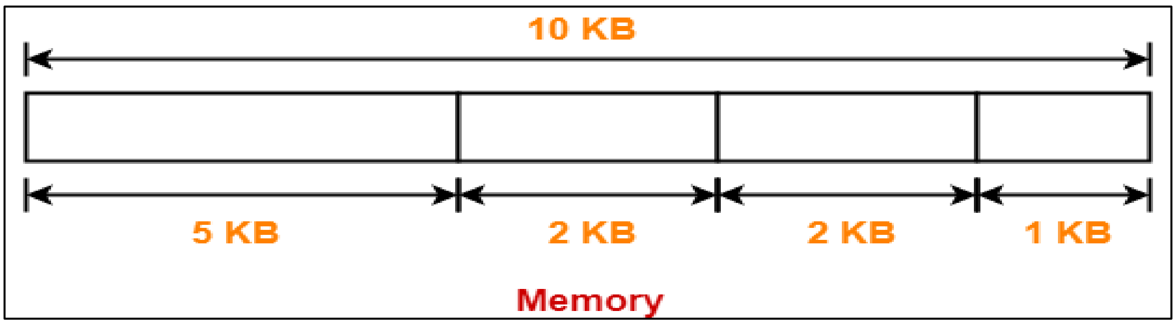
1. Static Partitioning
2. Dynamic Partitioning

Static Partitioning

- Static partitioning is a fixed size partitioning scheme.
- In this technique, main memory is pre-divided into fixed size partitions.
- The size of each partition is fixed and can not be changed.
- Each partition is allowed to store only one process.

Example

Under fixed size partitioning scheme, a memory of size 10 KB may be divided into fixed size partitions as-



- These partitions are allocated to the processes as they arrive.
- The partition allocated to the arrived process depends on the algorithm followed.

Advantages:

- Simple to implement.
- Easy to manage and design.

Disadvantages:

- Internal Fragmentation.
- Limitation on the size of process.
- External Fragmentation.
- Degree of Multiprogramming is less.

Dynamic Partitioning

- Dynamic partitioning is a variable size partitioning scheme.
- It performs the allocation dynamically.
- When a process arrives, a partition of size equal to the size of process is created.
- Then, that partition is allocated to the process.

Advantages:

- No Internal Fragmentation.
- Degree of Multiprogramming is dynamic.
- No limitation on the size of process.

Disadvantages:

- External fragmentation.
- Difficult implementation.

Internal Fragmentation

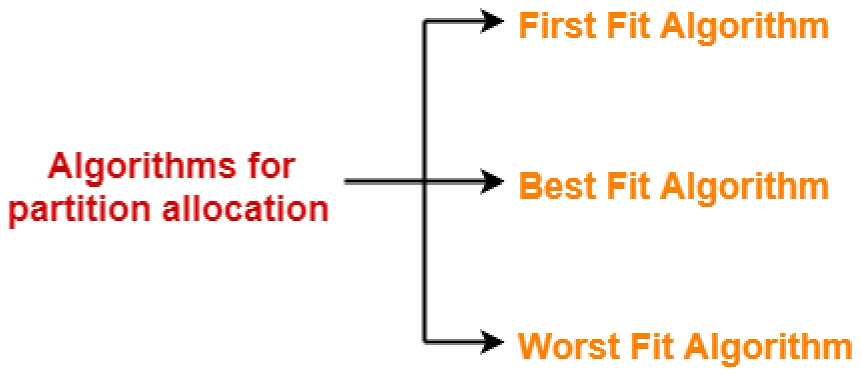
Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

External Fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used

Algorithms for Partition Allocation

Popular algorithms used for allocating the partitions to the arriving processes are



1. First Fit Algorithm
2. Best Fit Algorithm
3. Worst Fit Algorithm

1. First Fit Algorithm

- This algorithm starts scanning the partitions serially from the starting.
- When an empty partition that is big enough to store the process is found, it is allocated to the process.
- Obviously, the partition size has to be greater than or at least equal to the process size.

2. Best Fit Algorithm

- This algorithm first scans all the empty partitions.
- It then allocates the smallest size partition to the process.

3. Worst Fit Algorithm

- This algorithm first scans all the empty partitions.
- It then allocates the largest size partition to the process.

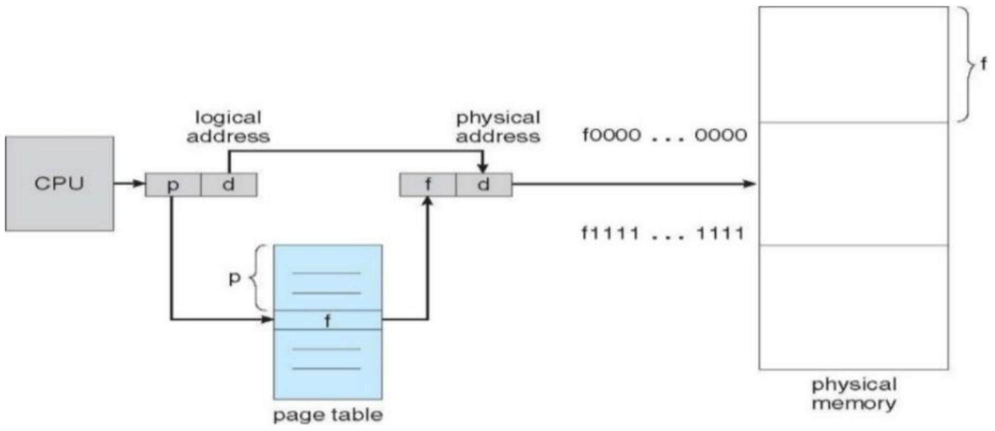
2)Explain the paging, segmentation concept with an example .Differentiate paging and segmentation.

Ans. Paging and segmentation are processes by which data is stored to, then retrieved from, a computer's storage disk.

Paging

- Main memory is divided into a number of equal-size blocks, are called frames.
- Each process is divided into a number of equal-size block of the same length as frames, are called Pages.
- A process is loaded by loading all of its pages into available frames.

Process of Translation from logical to physical addresses



⇒ Every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d). The page number is used as an index into a page table.

p	d

Logical address

⇒ The page table contains the base address of each page in physical memory. This base Address(f) is combined with the page offset(d) to define the physical memory address that is sent to the memory unit.

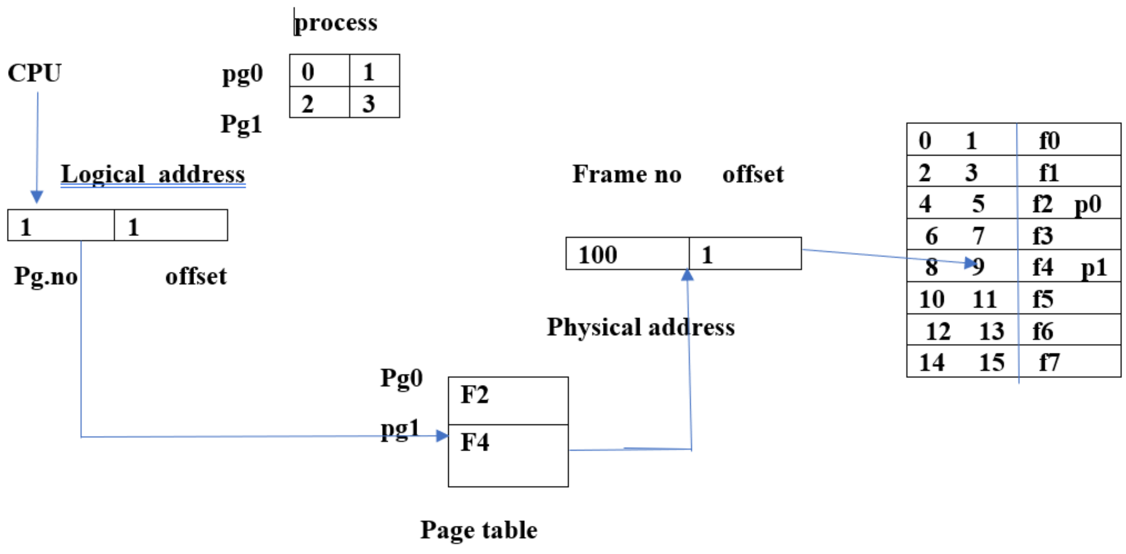
f	d

Physical address

⇒ If the size of logical-address space is 2^m and a page size is 2^n addressing units (bytes or words), then the high-order (m – n) bits of a logical address designate the page number and the n low-order bits designate the page offset. Thus, the logical address is as follows:

page number	**page offset**
p	**d**
m - n	**n**

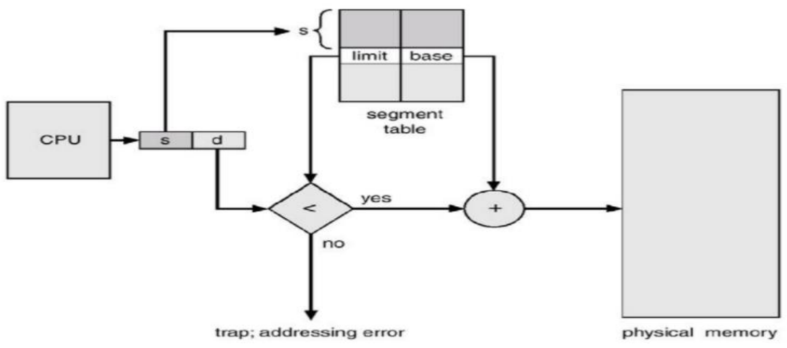
Example CPU requested for 3B **physical memory**



The logical address 3 is mapped to physical address 9.

Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
- A segment is a logical unit such as:
  main program, procedure ,function, method, object, local variables, globalvariables, common block, stack, symbol table, arrays



Segmentation Architecture

→ Logical address consists of a two tuple:

<segment-number, offset>

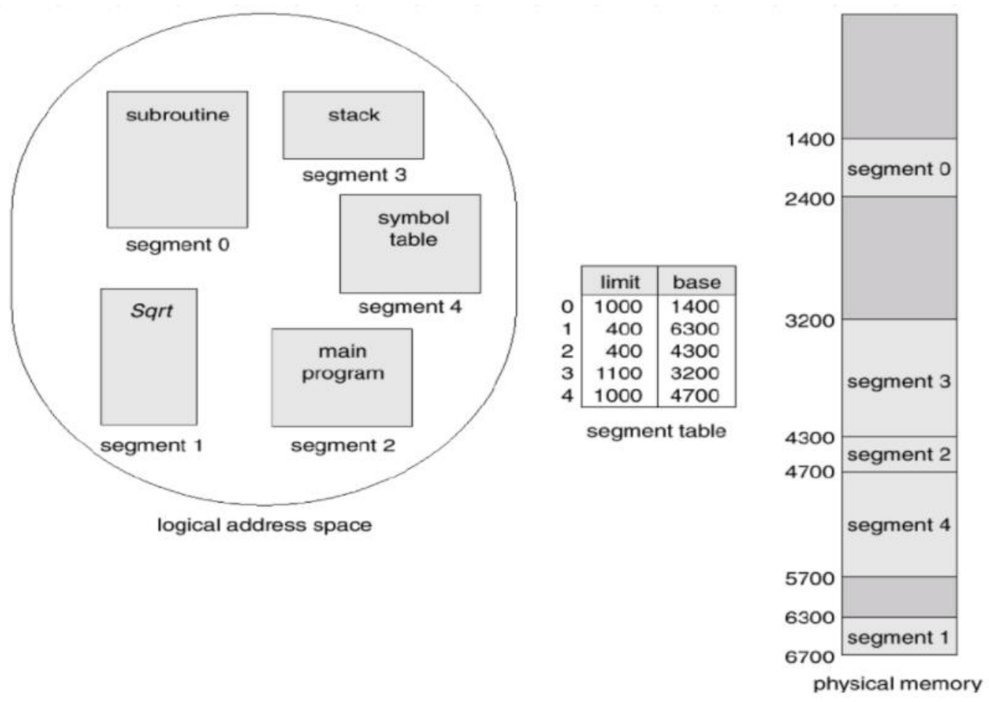
→**Segment table** – maps two-dimensional physical addresses; each table entry has:

base – contains the starting physical address where the segments reside in memory
limit – specifies the length of the segment

→Segment-table base register (STBR) points to the segment table's location in Memory.
→Segment-table length register (STLR) indicates number of segments used by program.

segment number s is legal if s<STLR

Example



Difference between paging and segmentation

Sr. No.	Key	Paging	Segmentation
1	Memory Size	In Paging, a process address space is broken into fixed sized blocks called pages.	In Segmentation, a process address space is broken in varying sized blocks called sections.
2	Accountability	Operating System divides the memory into pages.	Compiler is responsible to calculate the segment size, the virtual address and actual address.
3	Size	Page size is determined by available memory.	Section size is determined by the user.

Sr. No.	Key	Paging	Segmentation
4	Speed	Paging technique is faster in terms of memory access.	Segmentation is slower than paging.
5	Fragmentation	Paging can cause internal fragmentation as some pages may go underutilized.	Segmentation can cause external fragmentation as some memory block may not be used at all.
6	Logical Address	During paging, a logical address is divided into page number and page offset.	During segmentation, a logical address is divided into section number and section offset.
7	Table	During paging, a logical address is divided into page number and page offset.	During segmentation, a logical address is divided into section number and section offset.
8	Data Storage	Page table stores the page data.	Segmentation table stores the segmentation data.

Q3. Define Virtual memory, demand paging and page fault?

What are the steps required to handle page fault?

A: Virtual Memory:

Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory. In Virtual memory, the user can store processes with a bigger size than the available main memory.

Therefore, instead of loading one long process in the main memory, the OS loads the various parts of more than one process in the main memory. Virtual memory is mostly implemented with demand paging and demand segmentation.

Demand Paging:

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging. Demand paging is a type of swapping done in virtual memory. Generally, processes reside on Secondary memory. When we want to execute it, we swap it into Main memory. The data will not be copied when the data is already available on the memory. This is otherwise called a lazy evaluation because only the demanded pages of memory are being swapped from the secondary storage (disk space) to the main memory.

Page Fault:

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

Steps required to handle Page fault

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU.)

4)Explain what is page replacement algorithms FIFO , optimal , MRU , LRU .What is Belady’s Anamoly?

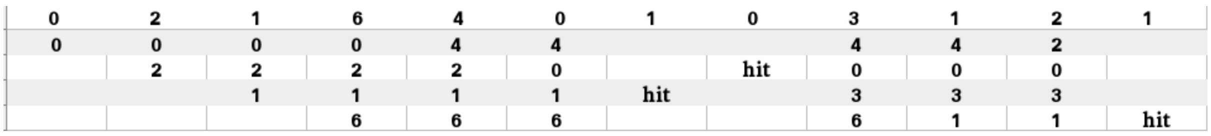
Ans. Whenever there is a page fault, page replacement is to be done. Then comes into play the page replacement algorithms.

Page Replacement Algorithms: The page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated.

1)First In First Out (FIFO) page replacement algorithm

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced, page in the front of the queue is selected for removal.

Example:



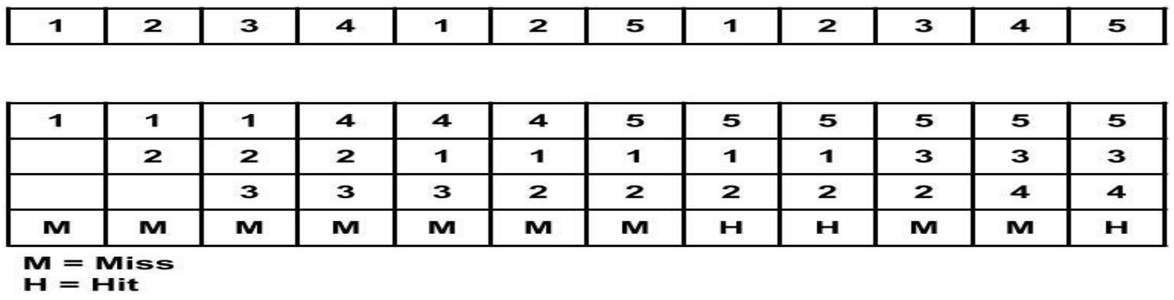
0	2	1	6	4	0	1	0	3	1	2	1
0	0	0	0	4	4			4	4	2	
	2	2	2	2	0		hit	0	0	0	
		1	1	1	1	hit		3	3	3	
			6	6	6			6	1	1	hit

Belady’s Anamoly

Belady’s anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns .

This phenomenon is commonly experienced when using the first-in first-out page replacement algorithm.

Example

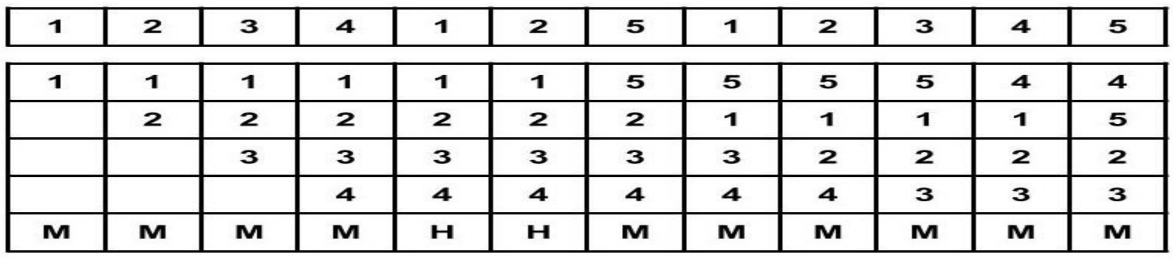


1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
M	M	M	M	M	M	M	H	H	M	M	H

M = Miss
H = Hit

Total Page Fault = 9

On increasing Number of frames



1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
M	M	M	M	H	H	M	M	M	M	M	M

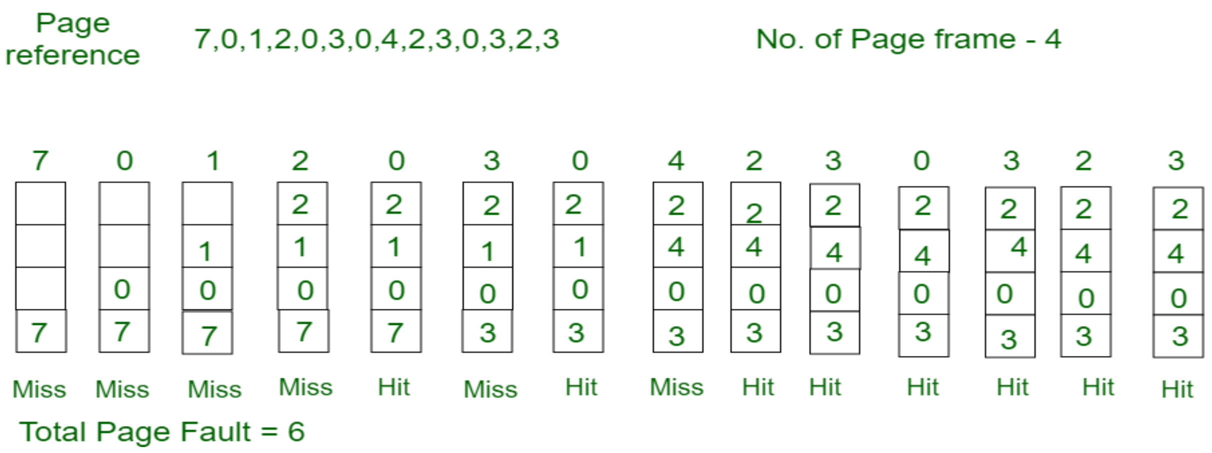
Total Page Fault = 10

2.)Optimal Page Replacement algorithm:

The Optimal page-replacement algorithm has the most reduced page-fault rate among all other page replacement algorithms and it will never suffer from the effect of Belady’s anomaly.

This algorithm replaces the page that will not be used for the longest period of time(in future).

Example



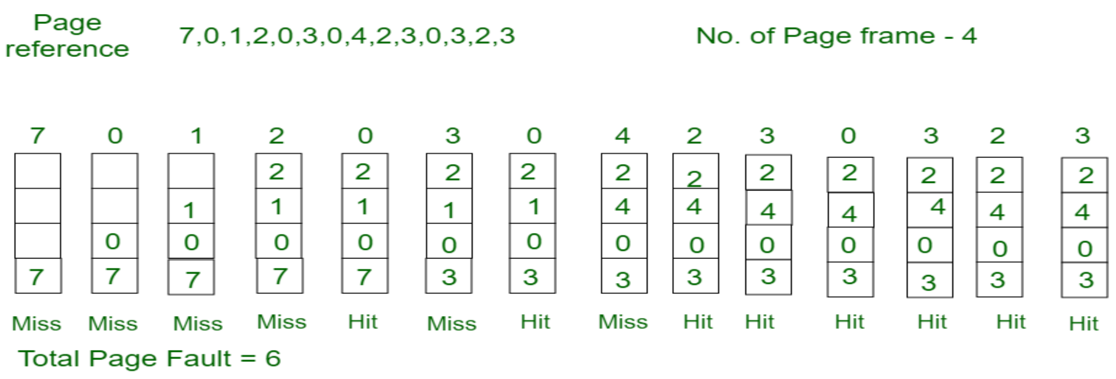
7,0,1,2-They are allocated to the empty slots —> **4 Page faults**
0 -page hit—> **0 Page fault.**
3- page miss- will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 -page hit—> **0 Page fault.**.
4 -page miss-will take the place of 1 as it is not used in future —> **1 Page Fault.**

For the rest of the sequence there is no page fault as all are available.

3) LRU(Least recently used) Page Replacement algorithm

LRU follows the concept of locality of reference as a base for its page replacement decisions.

It uses the strategy of replacing the page that has been unused for long time.



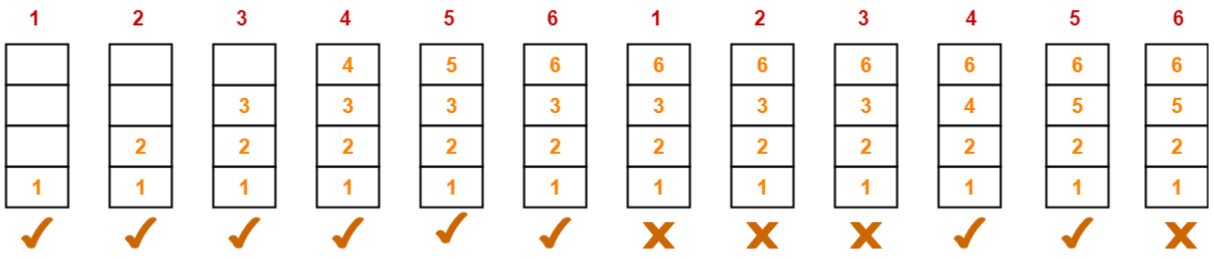
7,0,1,2-They are allocated to the empty slots —> **4 Page faults**
0 -page hit—> **0 Page fault.**

3 -page miss-it will take the place of 7 because it is least recently used —>**1 Page fault**
0 -page hit —> **0 Page fault**.
4 -page miss-will takes place of 1 as it is least recently used among others in the frame —> **1 Page Fault**
For the rest of the search there is no page fault as all are available.

4) MRU(Most recently used) Page Replacement algorithm

MRU replace the page which was just recently used in the frame of the main memory.

- Most recently used page will be required after the longest time.



1,2,3,4- They are allocated to the empty slots —> **4 Page faults**

5-page miss-replaced with most recently used 4

6-page miss-replaced with most recently used 5

1,2,3-page hit

4-page miss-replaced with most recently used 3

5-page miss-replaced with most recently used 4

6-page hit

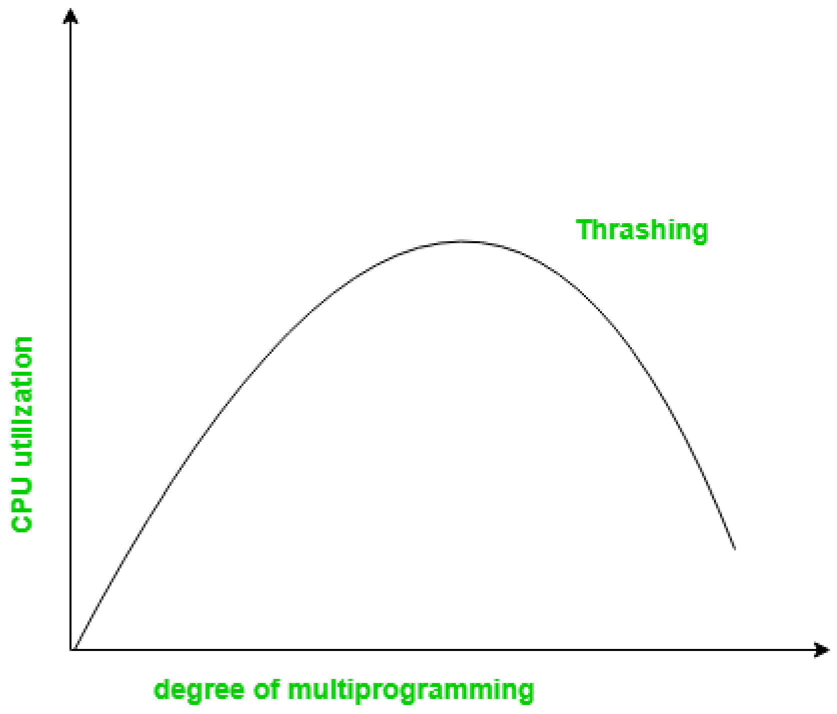
Q5. Explain about Thrashing. And discuss about Working Set Model.

A: Thrashing in Operating Systems:

When a program need space larger than RAM or it need space when RAM is full, Operating System will try to allocate space from secondary memory and behaves like it has that much amount of memory by serving to that program. This concept is called virtual memory. To know about thrashing we first need to know about page fault and swapping.

Page fault and swapping: We know every program divided into some pages. When a program needs a page which is not in RAM that is called page fault. Whenever a page fault happens, operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This is called swapping.

If this page fault and then swapping happening very frequently at higher rate, then operating system has to spend more time to swap these pages. This state is called thrashing. Because of this, CPU utilization is going to be reduced.



Thrash Handling Techniques

Thrashing need to be handled using some techniques. One of the thrash handling techniques is Working Set Model.

Working set Model

This model is based on locality. What locality is saying, the page used recently can be used again and also the pages which are nearby this page will also be used. Working set means set of pages in the most recent D time. The page which completed its D amount of time in working set automatically dropped from it. So, accuracy of working set depends on D we have chosen. This working set model avoid thrashing while keeping the degree of multiprogramming as high as possible.

UNIT -4

File System, System Implementation,

Mass-Storage Structure

Q.1.Explain the directory structure and file allocation methods in detail?

Directory Structure:

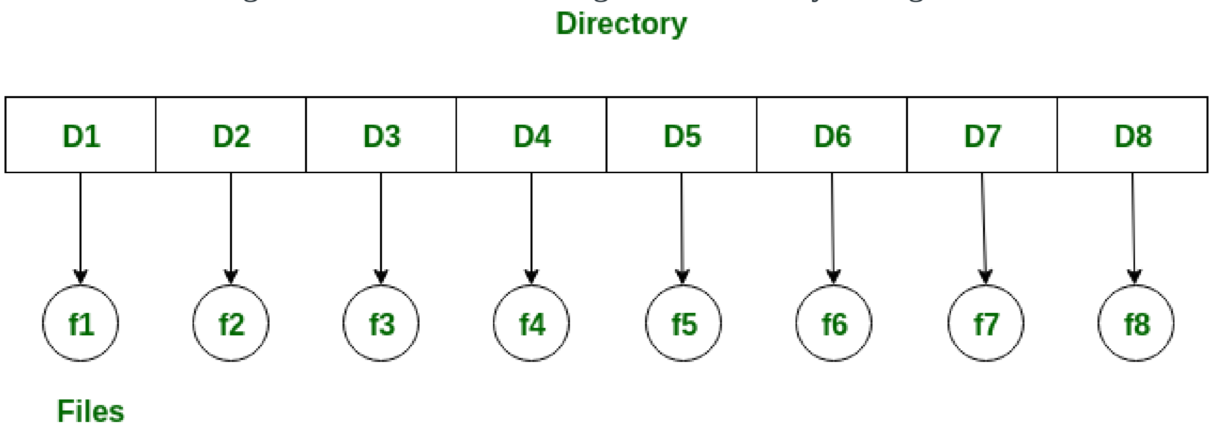
A **directory** is a container that is used to contain folders and files. It organizes files and folders.

Types of directory structures:

1. Single-Level Directory
2. Two-Level Directory
3. Tree-structured Directory
4. Acyclic-Graph Directory
5. General graph directory

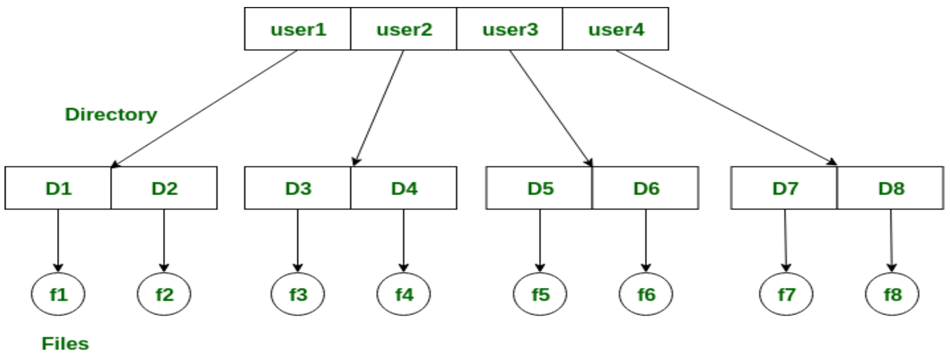
1.Single-Level Directory: The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand.

Limitations: when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names. Searching will become time taking if the directory is large.



2. Two-Level Directory: In the two-level directory structure, each user has its own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. Each UFD has a similar structure, but lists only the files of a single user.

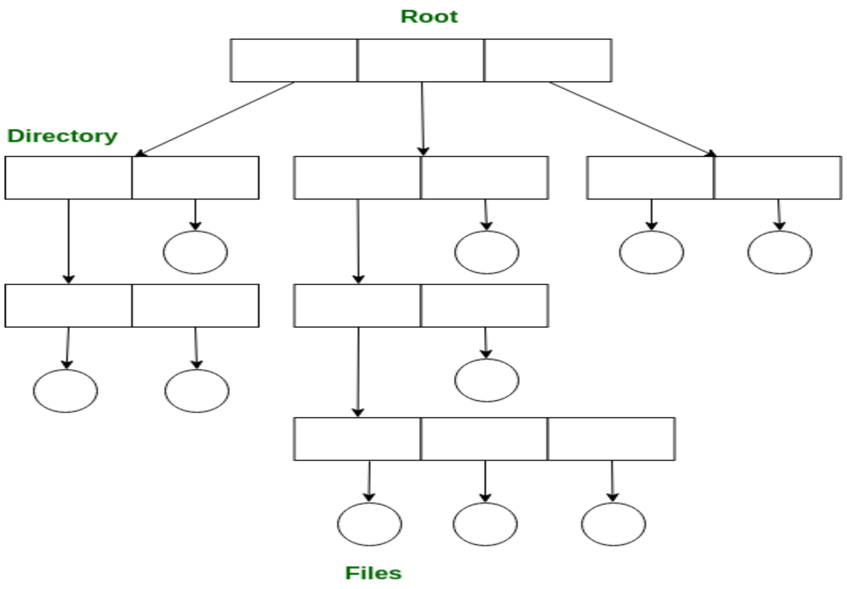
The disadvantages of the two-level directory are: In a two-level directory, one user cannot share the file with another user, it is not scalable.



3. Tree-structured directories: A tree structure is a more powerful and flexible approach to organize files and directories in hierarchical. There is a master directory, which has several user directories under it. Each of these user directories may have subdirectories and files as entries.

Disadvantages:

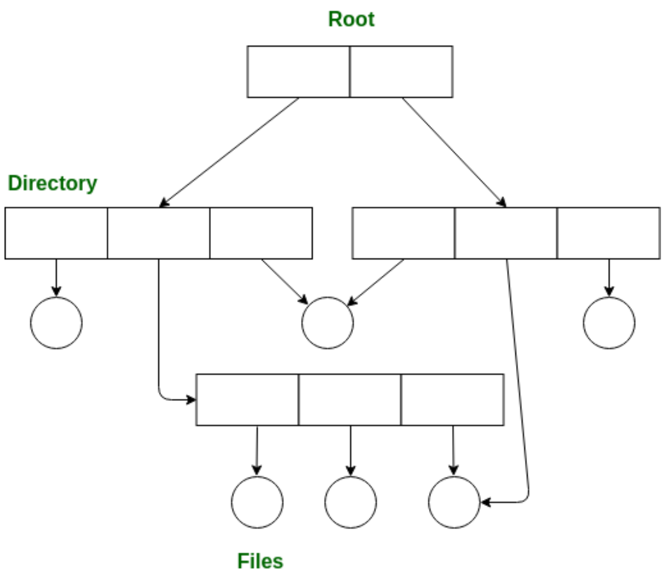
- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.



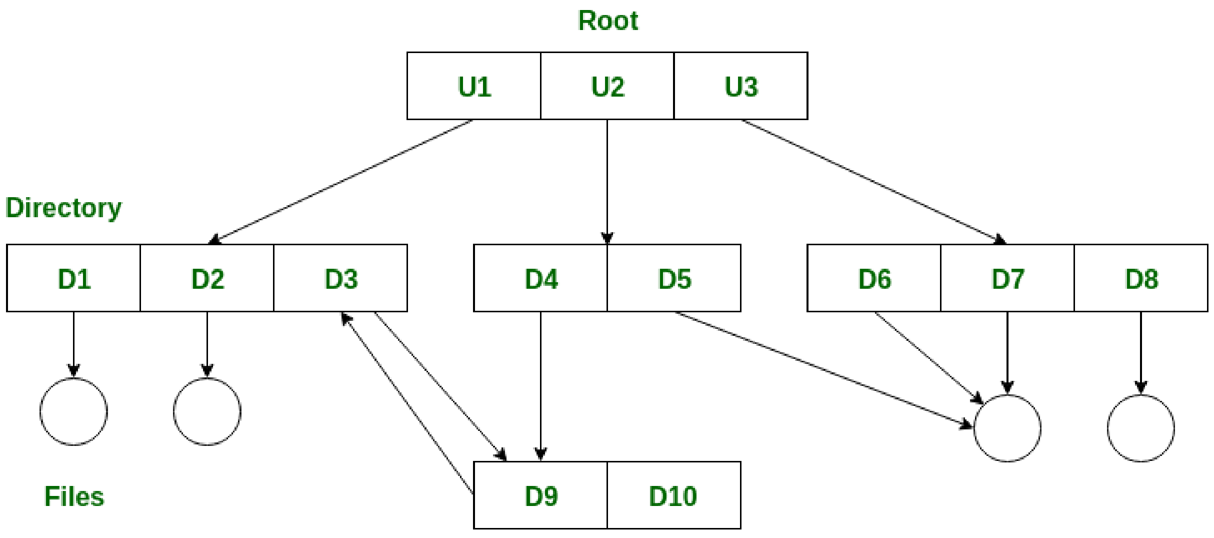
4. Acyclic-Graph Directories: An acyclic graph allows directories to have shared subdirectories and files. The same file or subdirectory may be in two different directories. An acyclic graph is a natural generalization of the tree structured directory scheme.

Disadvantages:

- We share the files via linking, in case deleting it may create the problem.
- In the case of a hard link, to delete a file we have to delete all the references associated with it.



5.General graph directory structure –
In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.
disadvantage
The main problem with this kind of directory structure is to calculate the total size or space that has been taken by the files and directories

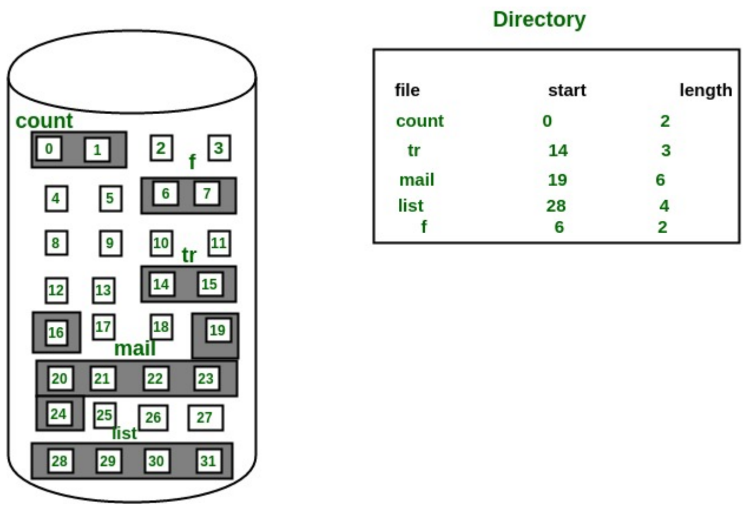


File Allocation Methods

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

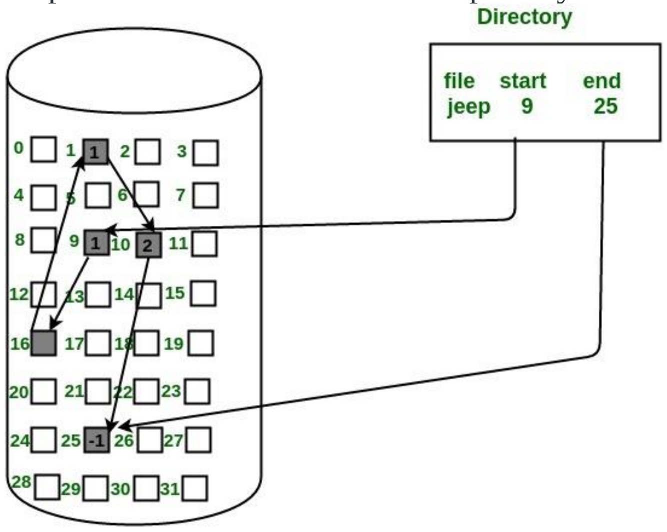
- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

1. Contiguous Allocation
In this scheme, each file occupies a contiguous set of blocks on the disk. we can determine the blocks occupied by the file given the starting block address and the length of the file.

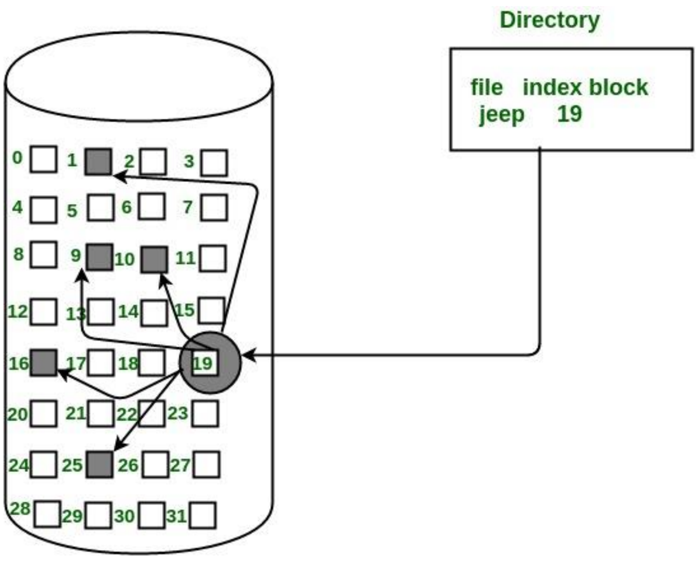


file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

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.



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.



Directory
file index block
jeep 19
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

Q.2. Explain the file access methods and free space management in detail?

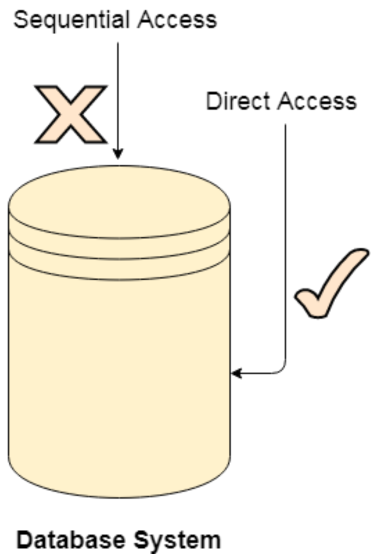
File Access Methods in Operating System:
There are three ways to access a file into a computer system:

 Sequential-Access
 Direct Access
 Index sequential Method

1.Sequential Access
It is the simplest access method. Data is accessed one file right after another file in an order.
In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file, then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.
Disadvantage: very slow access

2.Direct Access

The direct access is based on the disk model of a file since disk allows random access to any file block**.** we may read block 14 then block 59, and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.



Database System

3.Indexed Access
Index Access Method is another essential method of file, accessing. In this method, for a file an index is created. The index includes the pointer to the different blocks. If we want to find a record in the file, then first, we search in the index, and after that, with the help of the pointer, we can directly access the file.

In the Index Access method, we can search fast in the large database, and easy. But in this, the method needs some additional space to store the value of the index in the memory.

Free Space Management:
Bit Vector:
Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block.

Linked List:
In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory**.**

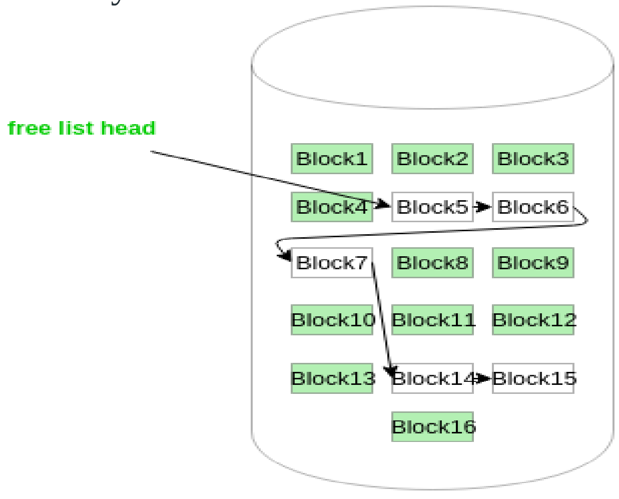


Figure - 2

Grouping:
This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are free, and the last block contains the address of next free n blocks.

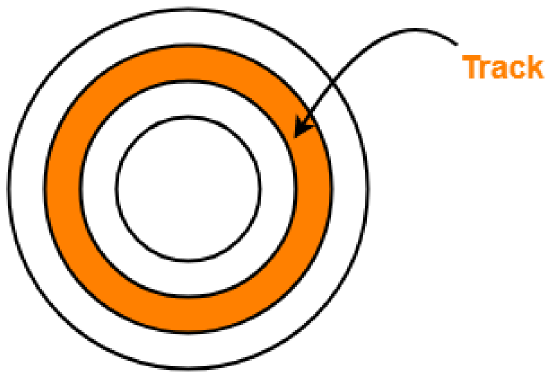
Counting –
This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.
Every entry in the list would contain:
1. Address of first free disk block
2. A number n

Q.3 Explain the Disk Architecture with diagram and discuss different parameters in detail?

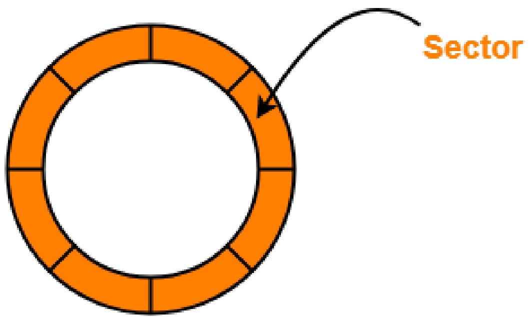
Disk Structure:

Magnetic disks provide the bulk of secondary storage for modern computer systems.

- The entire disk is divided into platters. disk platter has a flat circular shape, like a CD.
- Each platter consists of concentric circles called as tracks.
- These tracks are further divided into sectors which are the smallest divisions in the disk.

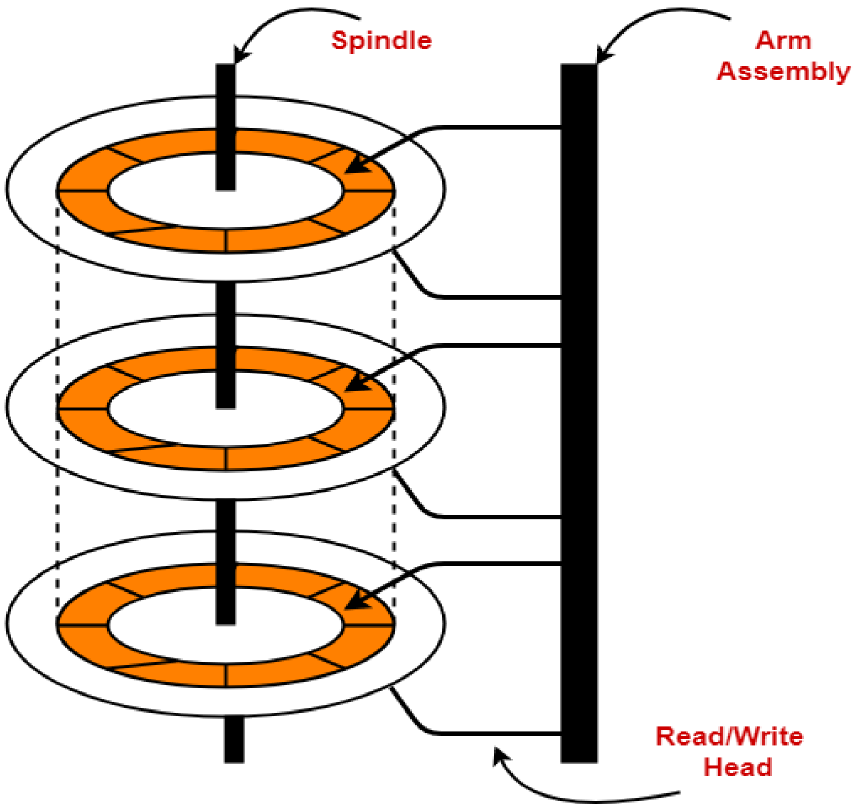


Disk divided into tracks



Track divided into sectors

- A cylinder is formed by combining the tracks at a given radius of a disk pack.
- There exists a mechanical arm called as Read / Write head.
- It is used to read from and write to the disk.
- Head must reach at a particular track and then wait for the rotation of the platter.
- The rotation causes the required sector of the track to come under the head.
- Each platter has 2 surfaces- top and bottom and both the surfaces are used to store the data.
- Each surface has its own read / write head.



Disk Performance Parameters:
The time taken by the disk to complete an I/O request is called as disk service time **or** disk access time**.**
Components that contribute to the service time are-

- Seek Time
- Rotational Latency
- Data Transfer Rate
- Controller Overhead

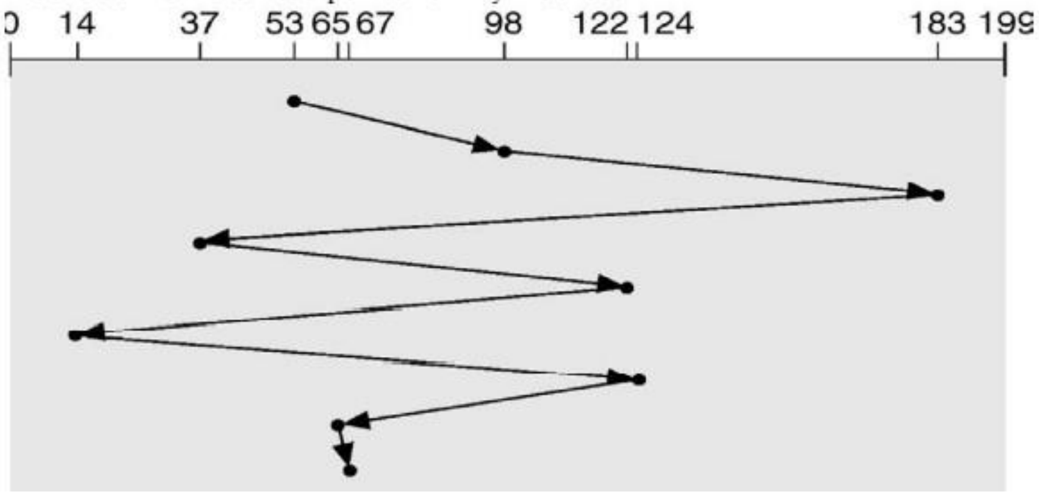
Capacity of Magnetic disks(C) = S x T x P x N Where S= no. of surfaces = 2 x no. of disks, T= no. of tracks in a surface, P= no. of sectors per track, N= size of each sector

Transfer Time: The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion: T = b / (r x N)
Where T= transfer time, b=number of bytes to be transferred,
N= number of bytes on a track, r= rotation speed, in revolutions per second.

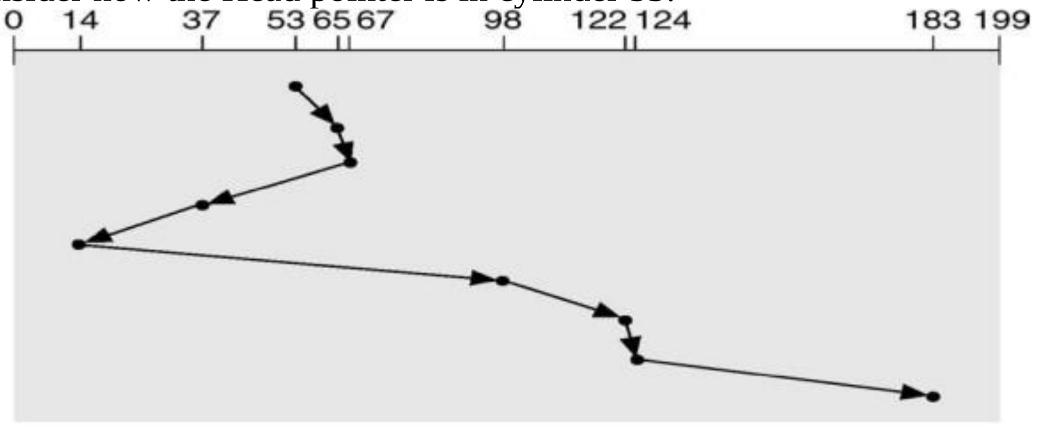
Q4. Explain Disk Scheduling Algorithms.

1.**FCFS Scheduling:** The simplest form of scheduling is first-in-first-serve(FIFS) scheduling, which processes items from the queue in sequential order.

We illustrate this with a request queue (0-199): 98, 183, 37, 122, 14, 124, 65, 67 Consider now the Head pointer is in cylinder 53 \.

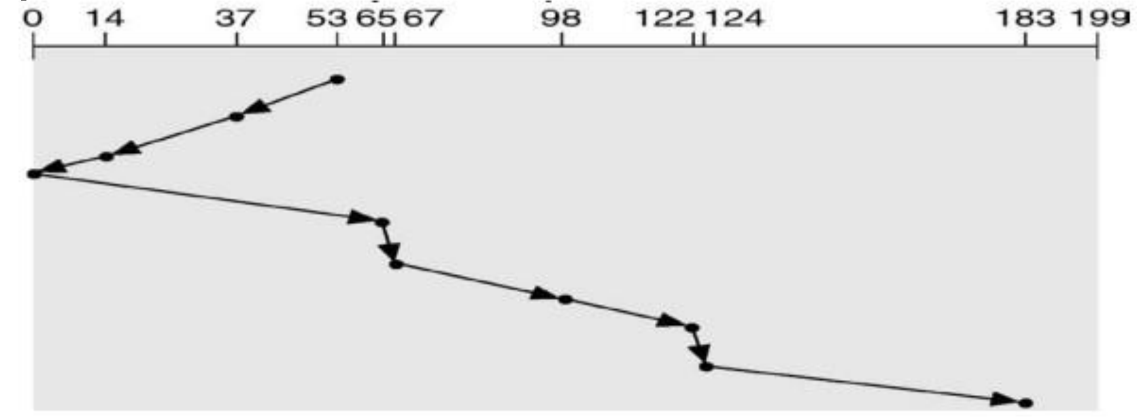


2. **SSTF:** It stands for shortest-seek-time-first (SSTF) algorithm. The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position. We illustrate this with a request queue (0-199): 98, 183, 37, 122, 14, 124, 65, 67 Consider now the Head pointer is in cylinder 53.

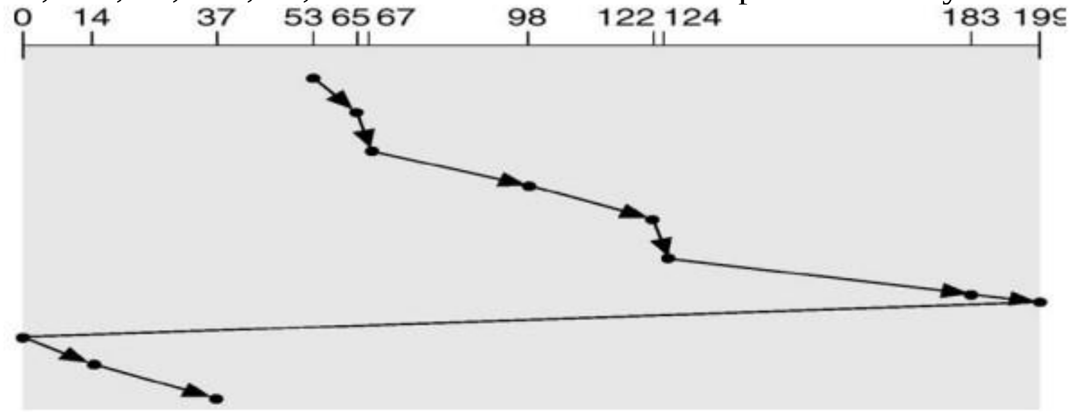


3.SCAN Scheduling: In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. We illustrate this with a request queue (0-

199): 98, 183, 37, 122, 14, 124, 65, 67 Consider now the Head pointer is in cylinder 53.

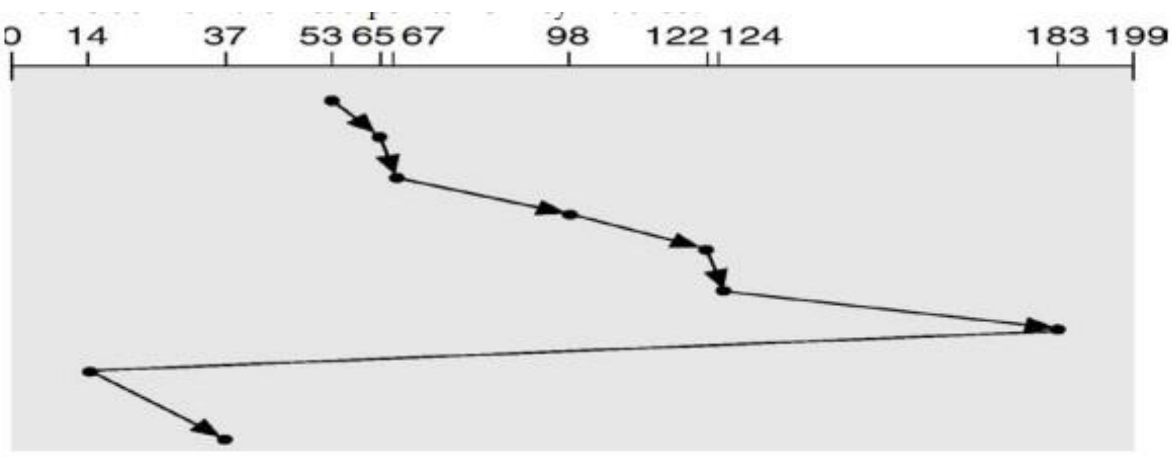


4.C-SCAN Scheduling: Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. The CSCAN scheduling algorithm
essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one. We illustrate this with a request queue (0-199): 98, 183, 37, 122, 14, 124, 65, 67. Consider now the Head pointer is in cylinder 53.



5.LOOK Scheduling and CLOOK: Practically, both SCAN and C-SCAN algorithm is not implemented this way. More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk. These versions of SCAN and C-SCAN are called LOOK and C-LOOK scheduling, because they look for a request before continuing to move in each direction. We illustrate this with a request queue (0-199): 98, 183, 37, 122, 14, 124, 65, 67 Consider now the Head pointer is in cylinder 53

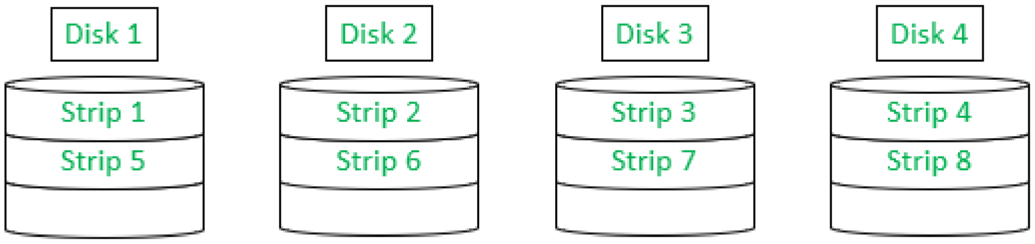
CLOOK


0
14
37
53
65
67
98
122
124
183
199

Q5. Explain the raid Structure in detail?

A Redundant Array of Inexpensive Disks (RAID) may be used to increase disk reliability. RAID may be implemented in hardware or in the operating system.

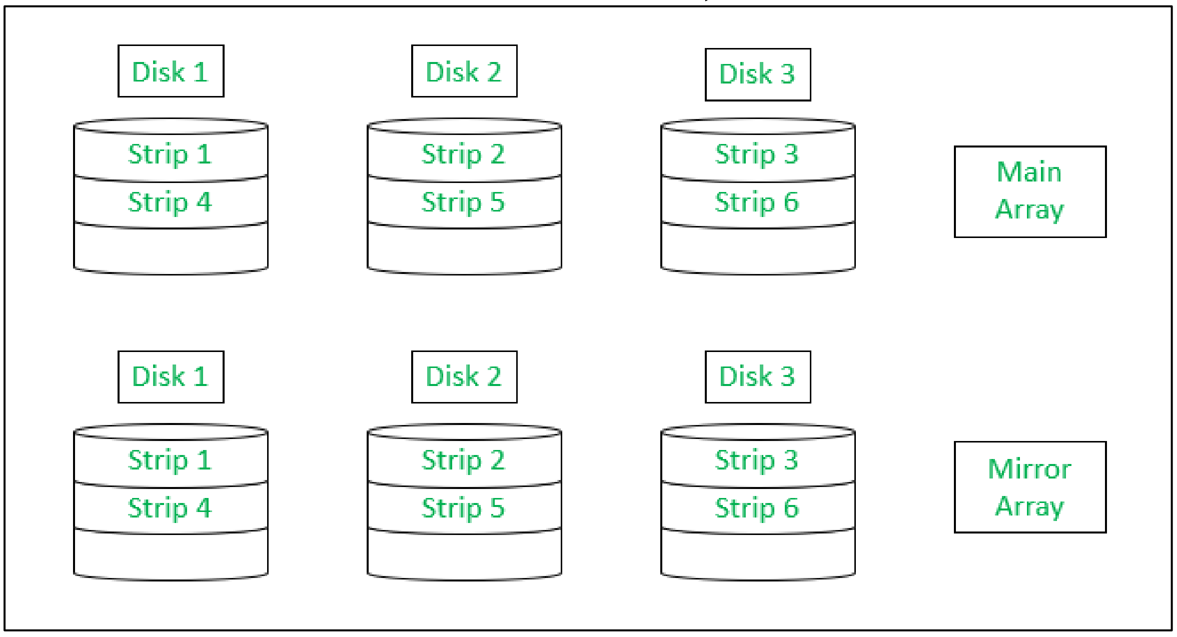
Raid 0(DATA STRIPPING):
RAID level 0 refers to disk arrays with striping at the level of blocks, but without any redundancy (such as parity bits)



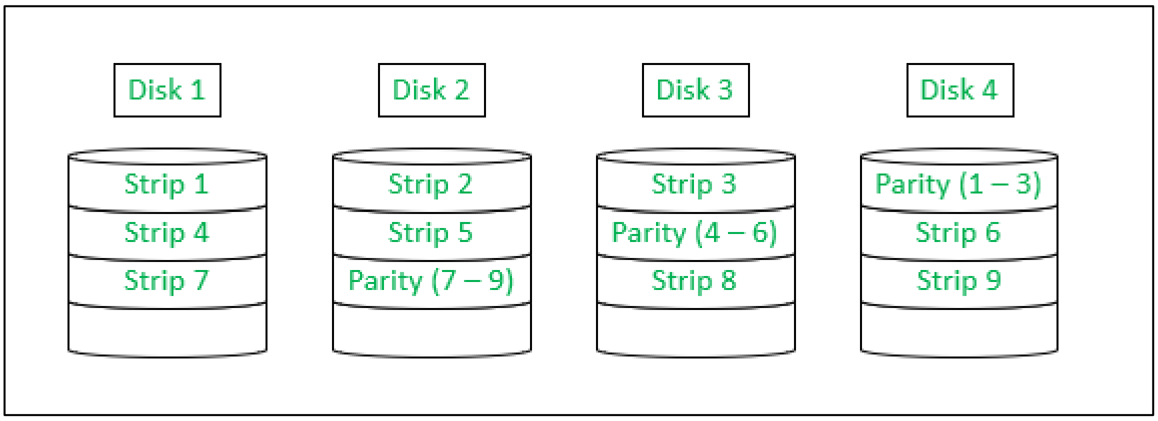
Striping is done at block level but without any redundancy. Writing performance is best in this level because due to absence of redundant information there is no need to update redundant information.

RAID Level 1:
RAID level 1 refers to disk mirroring.
In this level same data is copied on two different disks. It is the most expensive system. Because two copies of same data are available in two different disks, it allows parallel read.

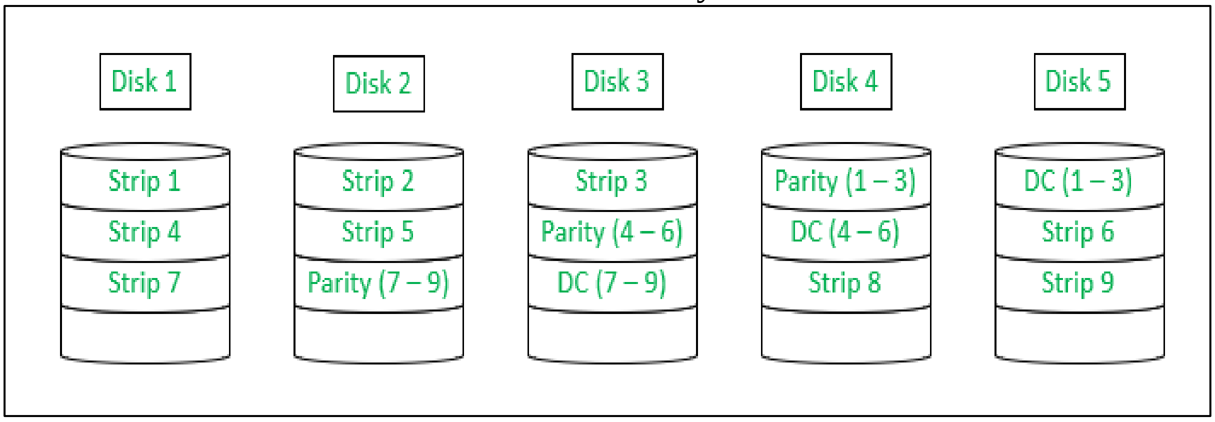
Level 1+0: Striping and Mirroring
This level is combination of level 0 and level 1, sometimes called as level 10.



RAID level 5: RAID level 5 or block-interleaved distributed parity
RAID level 5 distributes the parity block and data on all disks. For each block, one of the disks stores the parity and the others store data. RAID level 5 gives best performance for large read and write.



RAID Level 6: RAID level 6 (is also called the P+Q redundancy scheme) What happen if more than one disk fails at a time? This level stores extra redundant information to save the data against multiple disk failures. It does not use parity but uses Reed-Solomon codes for data recovery.



UNIT 5

I/O Systems, Protection and Security

QUESTIONS:

1. Explain DMA and kernel I/O subsystem in detail.
2. Define protection and security. What are goals and principles of protection? Explain the revocation of access rights?
3. What is domain structure? Explain access matrix with different variations?
4. Explain program threats, system threats and network threats?
5. Define cryptography? Explain cryptography techniques. Explain RSA algorithm with example.

1.Explain DMA and kernel I/O subsystem in detail.

Answer:

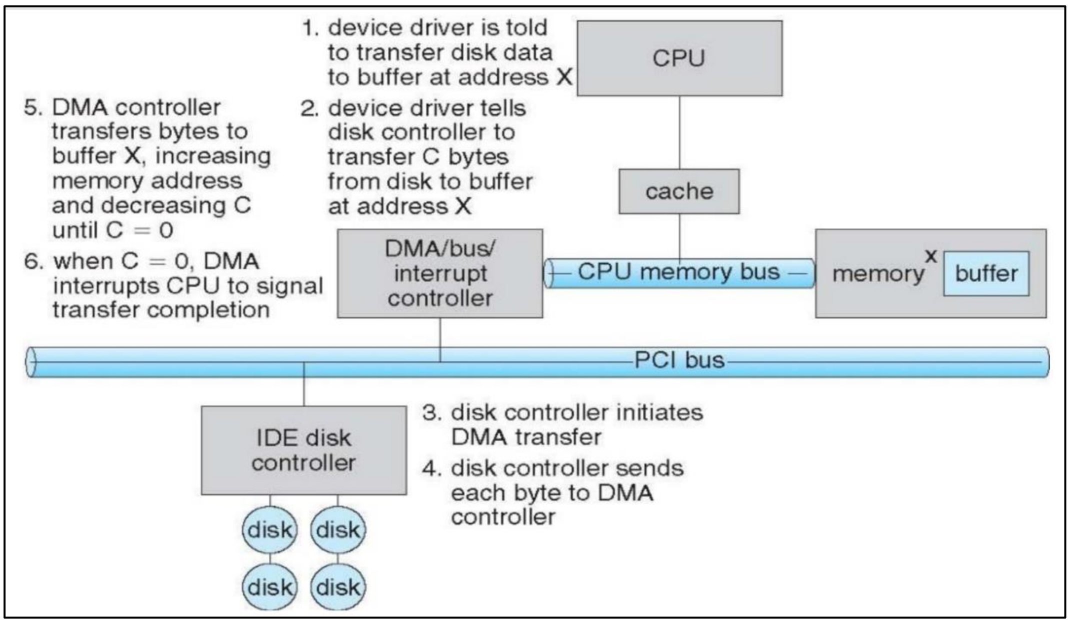
DMA (Direct Memory Access):

DMA is one of the techniques for performing I/O operations. A DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

The DMA units is capable of mimicking the Processor and taking over the control of the system bus just like a Processor. It needs to do this to transfer data to and from memory over the system bus. The DMA technique works as follows:

- ⇨ When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending to the DMA module the following information:
  - o Whether a read or write is requested, using the read or write control line between the processor and the DMA module.
  - o The address of the I/O device involved, communicated on the data lines.
  - o The starting location in memory to read from or write to, communicated on the data lines and stored by the DMA module in its address register.
  - o The number of words to be read or written, again communicated via the data lines and stored in the data count register.
- ⇨ Then the processor continues with other work. It has delegated this I/O operation to the DMA module.
- ⇨ The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor.
- ⇨ When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer.

Six Step Process to Perform DMA Transfer


1. device driver is told to transfer disk data to buffer at address X
CPU
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
cache
6. when C = 0, DMA interrupts CPU to signal transfer completion
DMA/bus/ interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
disk
disk
disk
disk

KERNEL I/O SUBSYSTEM:

The kernel provides many services related to I/O. Several services such as scheduling, caching, spooling, device reservation, and error handling – are provided by the kernel's I/O subsystem built on the hardware and device-driver infrastructure.

The I/O subsystem is also responsible for protecting itself from errant processes and malicious users.

1. **I/O Scheduling:**
   It is used to schedule a set of I/O requests means to determine a good order in which to execute them.
   Scheduling can improve the overall performance of the system, can share device access permission fairly to all the processes, reduce the average waiting time, response time, turnaround time for I/O to complete.
2. **Buffering:**
   A buffer is a memory area that stores data being transferred between two devices or between a device and an application. Buffering is done for three reasons.
   1. The first is to cope with a speed mismatch between producer and consumer of a data stream.

   2. The second use of buffering is to provide adaptation for data that have different data-transfer sizes.

   3. The third use of buffering is to support copy semantics for the application I/O.
3. **Caching:**
   A ***cache*** is a region of fast memory that holds a copy of data. Access to the cached copy is much easier than the original file. For instance, the instruction of the currently running process is stored on the disk, cached in physical memory, and copied again in the CPU's secondary and primary cache.
   The main difference between a buffer and a cache is that a buffer may hold only the existing copy of a data item, while a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

4. **Spooling and Device Reservation:**
   A ***spool*** is a buffer that holds the output of a device, such as a printer that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixes together.
   The OS solves this problem by preventing all output from continuing to the printer.
   The output of all applications is spooled in a separate disk file. When an application finish printing then the spooling system queues the corresponding spool file for output to the printer.

5. **Device Reservation:**
   It provides support for exclusive device access, by enabling a process to allocate an idle device, and to deallocate that device when it is no longer needed. Many operating systems provide functions that enable processes to coordinate exclusive access among them. It watches out for deadlock to avoid.

2. Define protection and security. What are goals and principles of protection? Explain the revocation of access rights?

Answer

Definitions:

- ⇨ **Protection:**
  - ➢ A mechanism that controls the access of programs, processes, or users to the resources defined by a computer system is referred to as protection.
  - ➢ Protection is a tool for multi-programming operating systems, allowing multiple users to safely share a common logical namespace, including a directory or files.
  - ➢ The protection of the system should confirm the approval of the process and users. Due to this, these licensed users and processes will care for the central processing unit, memory and alternative sources.
- ⇨ **Security:**
  - ➢ Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system.
  - ➢ The security systems covers the safety of their system resources (saved data, memory, disks, etc) across malignant alteration, illegal access, and disparity or inconsistency. The security gives a mechanism (authentication and encryption) to analyze the user to permit for using the system.

Goals and Principles of Protection:

⇨ **Goals of Protection**

- to prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies*, which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.
- Note that protection systems only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

⇨ **Principles of Protection**

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- Typically, each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

Revocation of access rights:

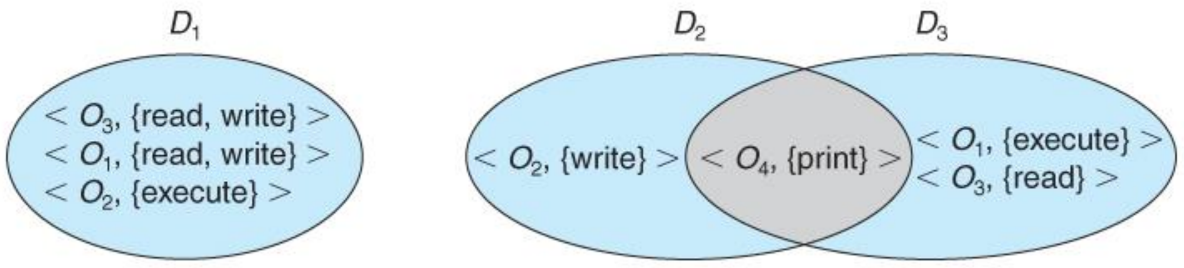
- Various options to remove the access right of a domain to an object
    - Immediate vs. delayed
    - Selective vs. general
    - Partial vs. total
    - Temporary vs. permanent
- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary, or permanent, as desired.
- With capabilities lists the problem is more complicated, because access rights are distributed throughout the system. A few schemes that have been developed include:
    - **Reacquisition -** Capabilities are periodically revoked from each domain, which must then re-acquire them.
    - **Back-pointers -** A list of pointers is maintained from each object to each capability which is held for that object.
    - **Indirection -** Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must then re-acquire access rights to continue.
    - **Keys -** A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process.
        - A master key is associated with each object.
        - When a capability is created, its key is set to the object's master key.
        - As long as the capability's key matches the object's key, then the capabilities remain valid.
        - The object master key can be changed with the set-key command, thereby invalidating all current capabilities.
        - More flexibility can be added to this scheme by implementing a ***list*** of keys for each object, possibly in a global table.

3. What is domain structure? Explain access matrix with different variations?

Answer

Domain Structure:

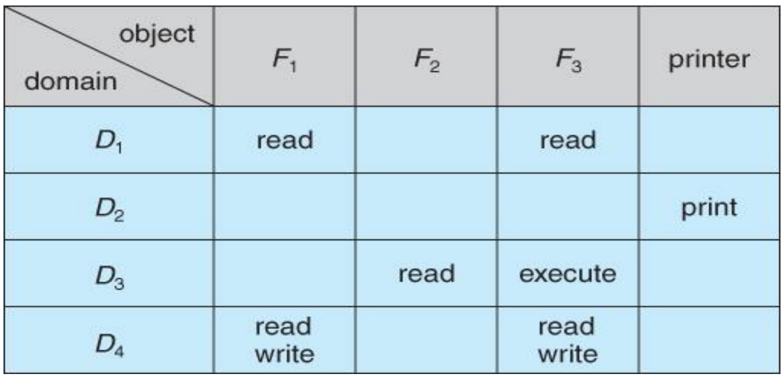
- A ***protection domain*** specifies the resources that a process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- An ***access right*** is the ability to execute an operation on an object.
- A domain is defined as a set of < object, {access right set}> pairs, as shown below. Note that some domains may be disjoint while others overlap.



- The association between a process and a domain may be *static* or *dynamic*.
  - If the association is static, then the need-to-know principle requires a way of changing the contents of the domain dynamically.
  - If the association is dynamic, then there needs to be a mechanism for ***domain switching.***
- Domains may be realized in different fashions - as users, or as processes, or as procedures. E.g.: if each user corresponds to a domain, then that domain defines the access of that user, and changing domains involves changing user ID.

Access Matrix

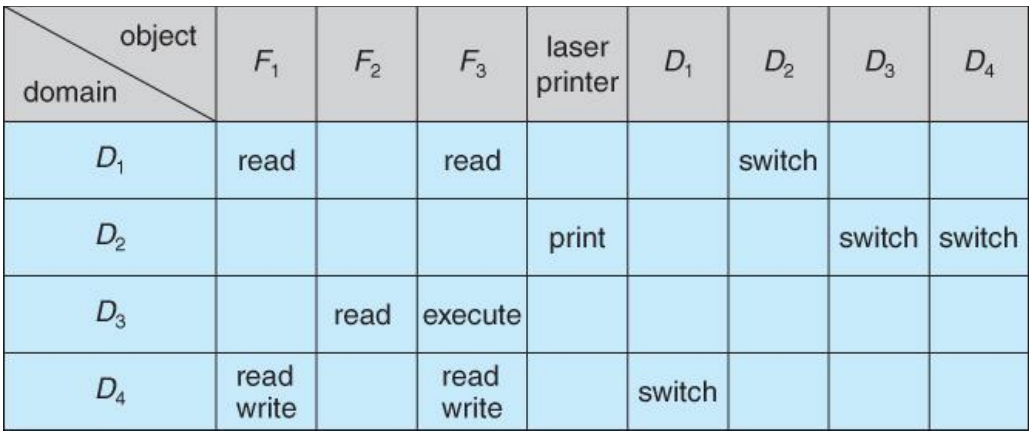
- The model of protection in which columns represent different system resources and rows represent different protection domains is referred to as Access Matrix. Entries within the matrix indicate what access that domain has to that resource.



object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure 14.3 - Access matrix.

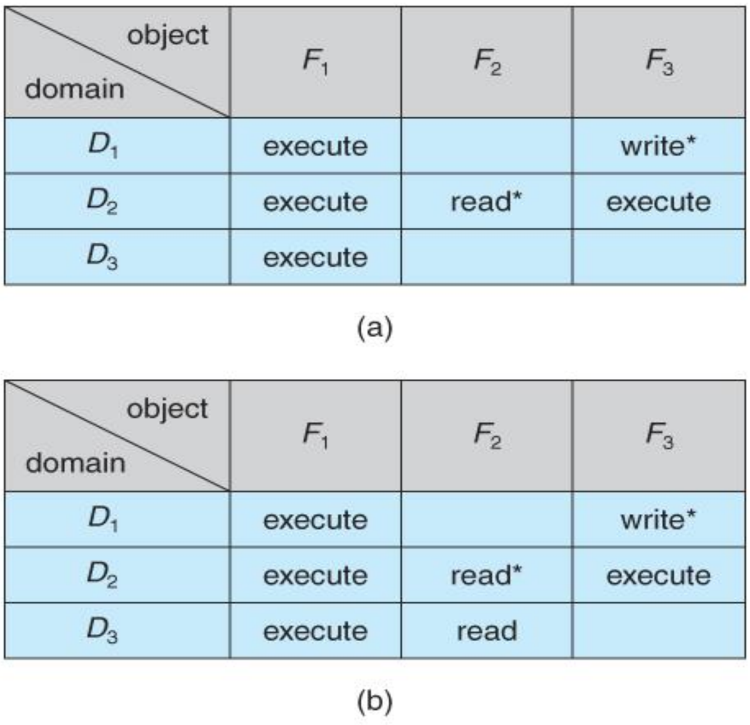
- Domain switching can be easily supported under this model, simply by providing "switch" access to other domains:



object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure 14.4 - Access matrix of Figure 14.3 with domains as objects.

- The ability to ***copy*** rights is denoted by an asterisk, indicating that processes in that domain have the right to copy that access within the same column, i.e., for the same object. There are two important variations:
    - If the asterisk is removed from the original access right, then the right is ***transferred,*** rather than being copied. This may be termed a ***transfer*** right as opposed to a ***copy*** right.
    - If only the right and not the asterisk is copied, then the access right is added to the new domain, but it may not be propagated further. That is the new domain does not also receive the right to copy the access. This may be termed a ***limited copy*** right, as shown in Figure below:



object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

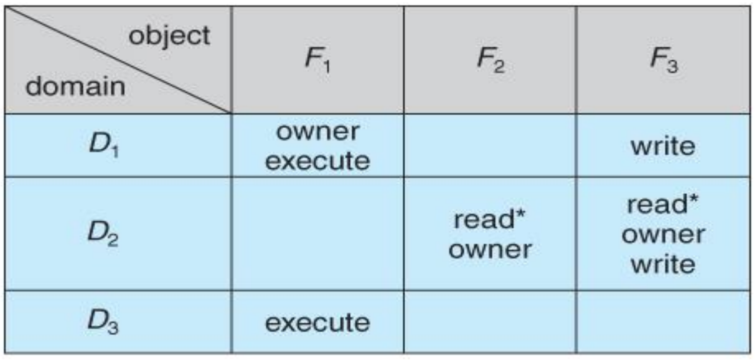
(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

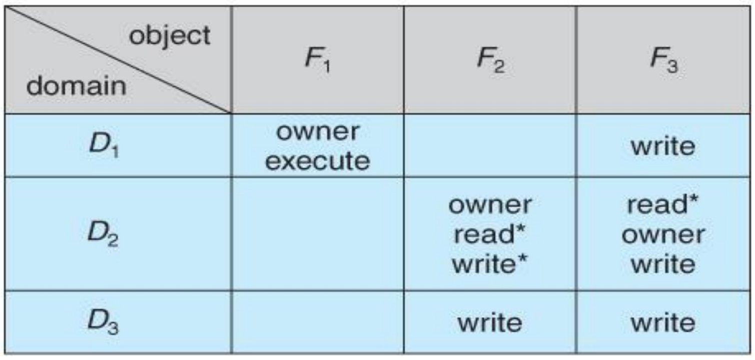
Figure 14.5 - Access matrix with *copy* rights.

- The ***owner*** right adds the privilege of adding new rights or removing existing ones:



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

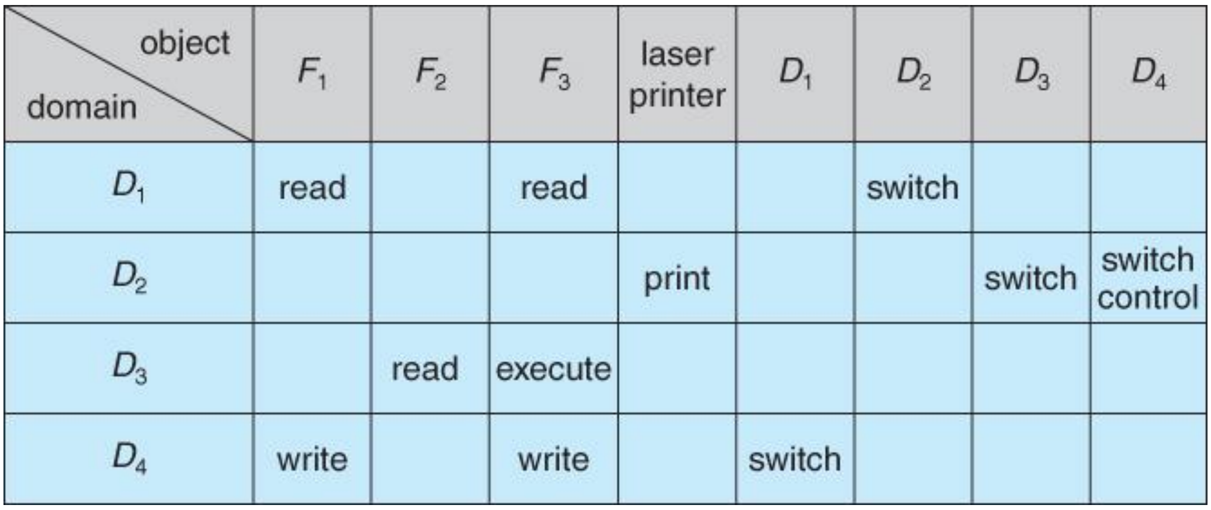


object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Figure 14.6 - Access matrix with *owner* rights.

- Copy and owner rights only allow the modification of rights within a column. The addition of ***control rights***, which only apply to domain objects, allow a process operating in one domain to affect the rights available in other domains. For example, in the table below, a process operating in domain D2 has the right to control any of the rights in domain D4.



object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Figure 14.7 - Modified access matrix of Figure 14.4

4. Explain program threats, system threats and network threats?

Answer

Threat: A program which has the potential to cause serious damage to the system. Threats can be classified into the following two categories:

1. Program Threats
2. System and Network Threats

- ⇨ **Program Threats:**
  A program written by a cracker to hijack the security or to change the behaviour of a normal process.

Types of Program Threats –

- ➢ **Virus:**
  An infamous threat, known most widely. It is a self-replicating and a malicious thread which attaches itself to a system file and then rapidly replicates itself, modifying and destroying essential files leading to a system breakdown.
  - Some types of computer viruses:
    - file/parasitic – appends itself to a file
    - boot/memory – infects the boot sector
    - source code – searches and modifies source codes
    - encrypted – encrypted virus + decrypting code
    - polymorphic – changes in copying each time
- ➢ **Trojan Horse:**
  - A standalone malicious program which may give full control of infected PC to another PC is called Trojan horse. It may make copies of them, harm the host computer systems, or steal information.
  - Trojans are designed as they can cause serious damage by deleting files and destroying information on your system.
  - Trojans allow confidential or personal information to be compromised by system creating a backdoor on your computer that gives unauthorized users access to your system.
  - Most popular Trojan horses are Beast, Zeus, The Blackhole Exploit Kit, Flashback Trojan, Netbus,Subseven, etc.,.
- ➢ Trap Door:
  - A trap door is kind of a secret entry point into a program that allows anyone gain access to any system without going through the usual security access procedures.
  - Other definition of trap door is it is a method of bypassing normal authentication methods. Therefore, it is also known as back door.
  - Programmers use Trap door legally to debug and test programs. Trap doors turns to threats when any dishonest programmers to gain illegal access.
- ➢ **Logic Bomb:**
  A logic bomb is a malicious piece of code that's secretly inserted into a computer network, operating system, or software application. It lies dormant until a specific condition occurs. When this condition is met, the logic bomb is triggered — devastating a system by corrupting data, deleting files, or clearing hard drives. Logic bombs are also sometimes referred to as **code bombs** and **cyber bombs**.

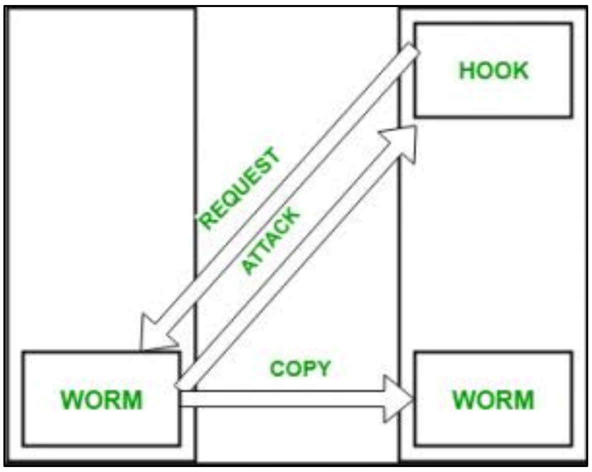
- ## System and Network Threats:

  System threats are described as the misuse of system services and network connections to cause user problems. These threats may be used to trigger the program threats over an entire network, known as program attacks. They are also used as a medium to launch program threats. These threats attack the operating system or the network itself, or leverage those systems to launch their attacks.

Types of System and Network Threats:

- **Worm:**

  A Worm is an infection program which spreads through networks. Unlike a virus, they target mainly LANs. A computer affected by a worm attacks the target system and writes a small program "hook" on it. This hook is further used to copy the worm to the target computer. This process repeats recursively, and soon enough all the systems of the LAN are affected. The basic functionality of a worm can be represented as:



- **Port Scanning:**

  Port Scanning is a method by which the cracker determines the system's vulnerabilities for an attack. It is a fully automated process that includes connecting to a specific port via TCP/IP. To protect the attacker's identity, port scanning attacks are launched through Zombie Systems, which previously independent systems now serve their owners while being utilized for such terrible purposes.
- **Denial of Service:**

  Denial of Service (DOS) attacks do not attempt to actually access or damage systems, but merely to clog them up so badly that they cannot be used for any useful work.

  Security systems that lock accounts after a certain number of failed login attempts are subject to DOS attacks which repeatedly attempt logins to all accounts with invalid passwords strictly in order to lock up all accounts.

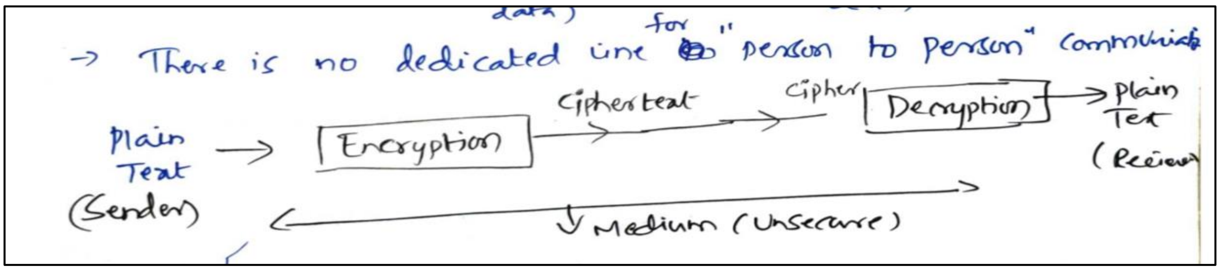
5.Define cryptography? Explain cryptography techniques. Explain RSA algorithm with example.

Answer

Cryptography:

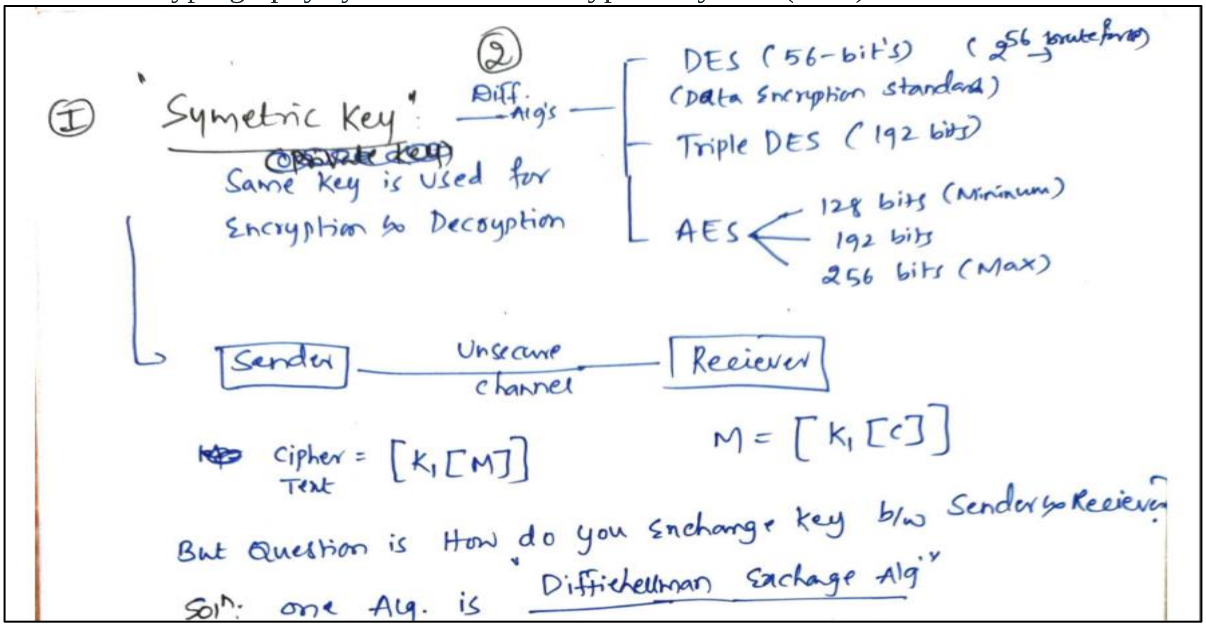
Cryptography is technique of securing information and communications through use of codes so that only those persons for whom the information is intended can understand it and process it. Thus, preventing unauthorized access to information. The prefix “crypt” means “hidden” and suffix “graphy” means “writing”.

Techniques used For Cryptography:
In today’s age of computers cryptography is often associated with the process where an ordinary plain text is converted to cipher text which is the text made such that intended receiver of the text can only decode it and hence this process is known as **encryption**. The process of conversion of cipher text to plain text this is known as **decryption**.



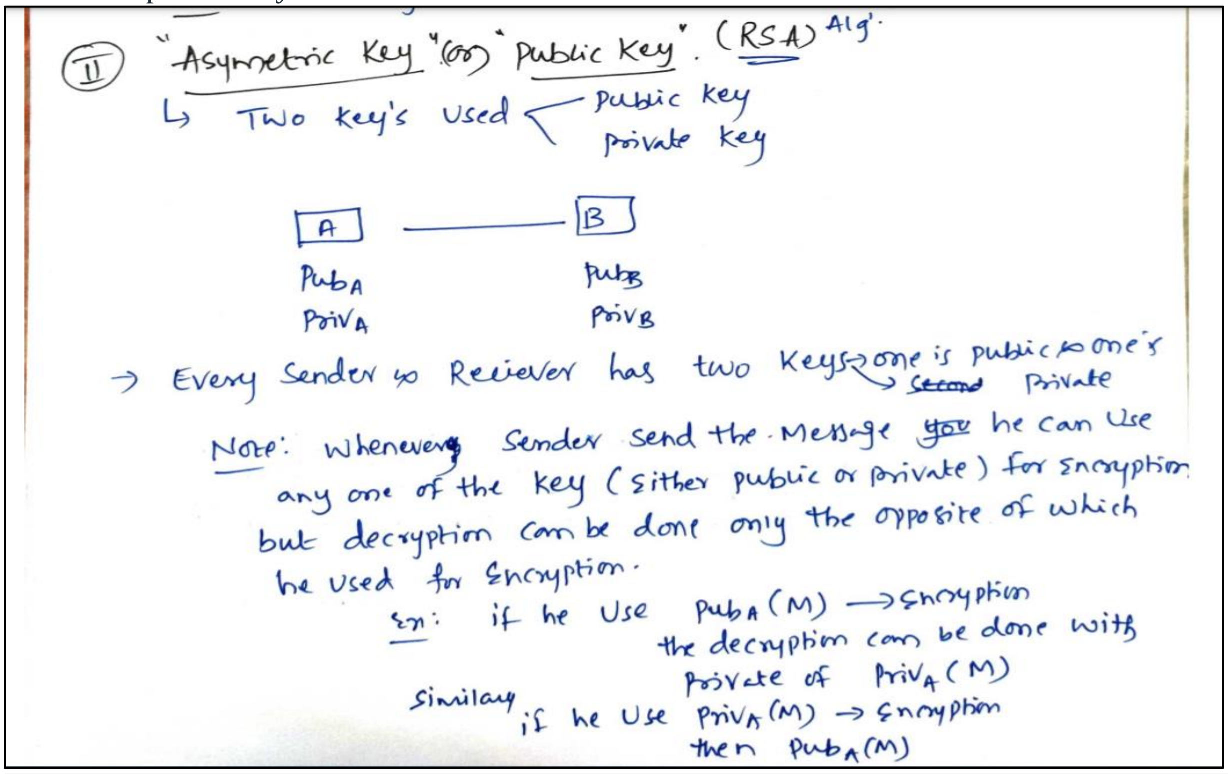
Types of Cryptography:

- **Symmetric Key Cryptography:**
  It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver have to somehow exchange key in a secure manner. The most popular symmetric key cryptography system is Data Encryption System (DES).



I
Symetric Key
Same key is used for Encryption & Decryption
2
Diff. Alg's
DES (56-bit's) (2^56 brute force)
(Data Encryption standard)
Triple DES (192 bits)
AES
128 bits (Minimum)
192 bits
256 bits (Max)
Sender
Unsecure channel
Reciever
Cipher Text = [K1 [M]]
M = [K1 [C]]
But Question is How do you exchange key b/w Sender & Reciever
Soln: one Alg. is Diffiehellman Exchange Alg"

- **Asymmetric Key Cryptography:**
  Under this system, a pair of keys is used to encrypt and decrypt information. A **public key** is used for encryption and a **private key** is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone knows the private key.



(II) "Asymmetric Key" (or) "Public Key". (RSA) Alg'.

↳ Two key's used < public key / private key

$\boxed{A}$ —— $\boxed{B}$

Pub_A $\quad$ Pub_B

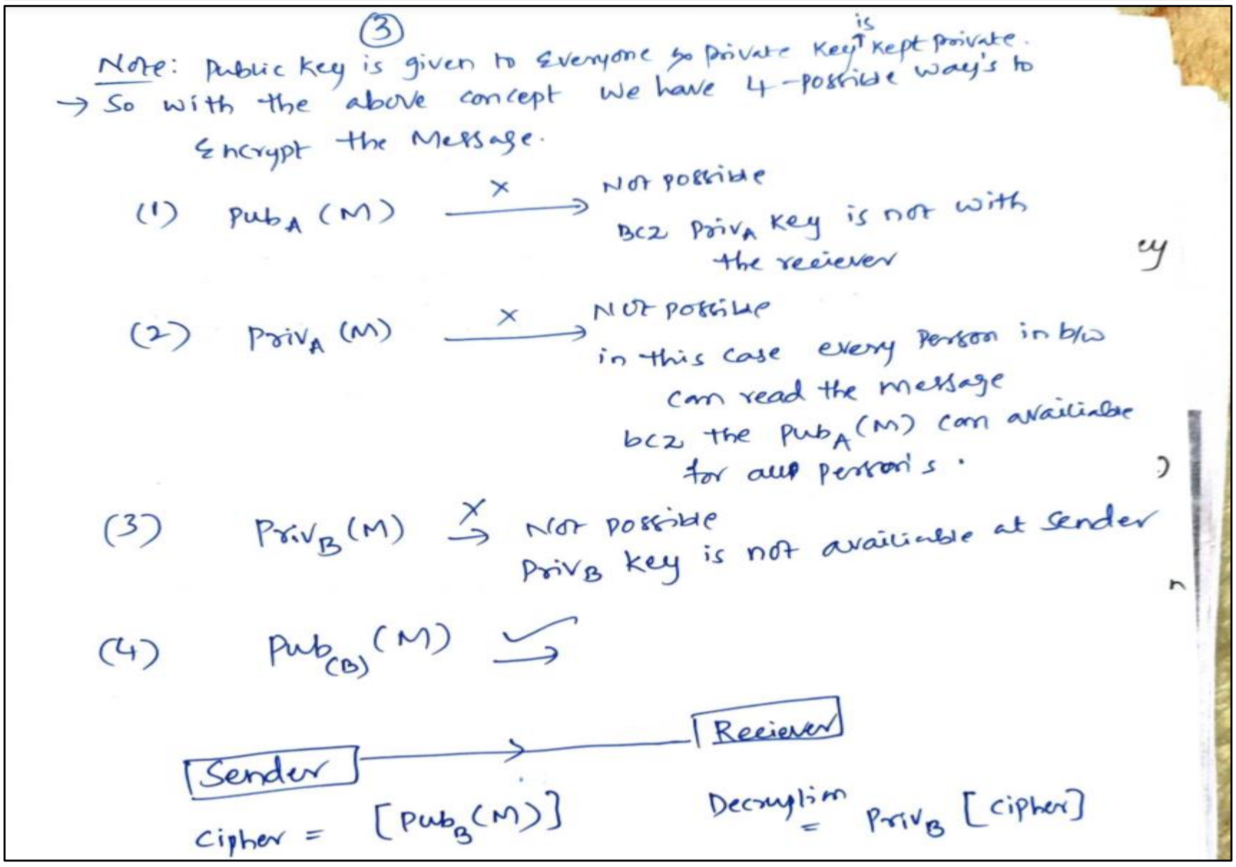
$Priv_A$ $\quad$ $Priv_B$

→ Every Sender & Reciever has two keys → one is public & one's ~~Second~~ Private

Note: whenever Sender send the Message ~~you~~ he can Use any one of the key (either public or private) for Encryption but decryption can be done only the opposite of which he used for Encryption.

Ex: if he Use $Pub_A(M) \rightarrow$ Encryption the decryption can be done with private of $Priv_A(M)$

Similary if he Use $Priv_A(M) \rightarrow$ Encryption then $Pub_A(M)$



(3)

Note: Public key is given to Everyone & Private Key is kept private.

→ So with the above concept we have 4-possible way's to Encrypt the Message.

(1) $Pub_A(M) \xrightarrow{\times}$ Not possible bcz $Priv_A$ key is not with the reciever

(2) $Priv_A(M) \xrightarrow{\times}$ Not possible in this case every Person in b/w can read the message bcz the $Pub_A(M)$ can available for all person's.

(3) $Priv_B(M) \xrightarrow{\times}$ Not possible $Priv_B$ key is not availiable at sender

(4) $Pub_{(B)}(M) \xrightarrow{\checkmark}$

$\boxed{Sender} \longrightarrow \boxed{Reciever}$

Cipher = $[Pub_B(M)]$ $\quad$ Decryption = $Priv_B$ [cipher]

RSA Algorithm:

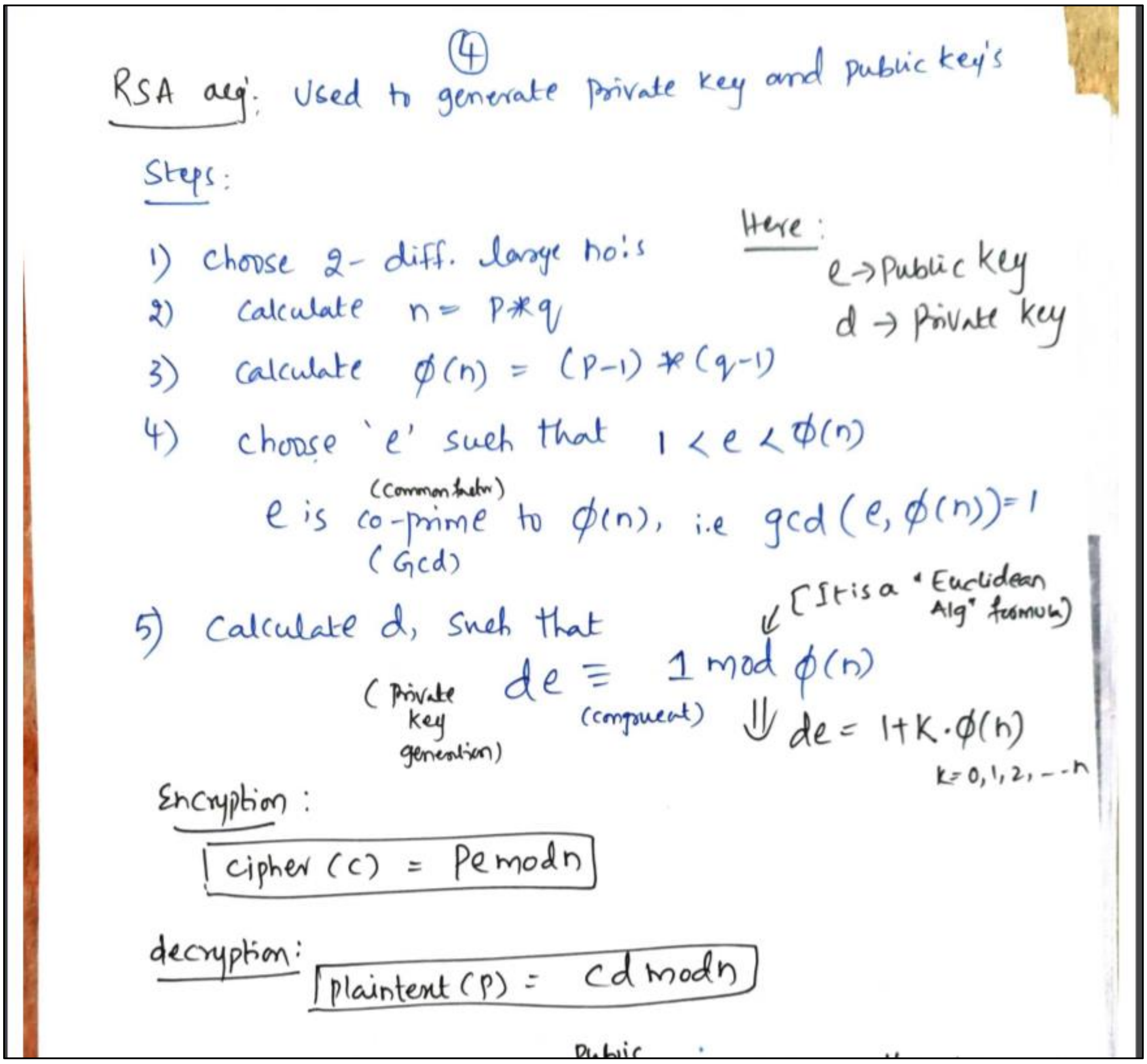
RSA algorithm is asymmetric cryptography algorithm and is considered as the most secure way of encryption.

The acronym "RSA" comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977.

Asymmetric actually means that it works on two different keys i.e., **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

The RSA algorithm holds the following features −

- RSA algorithm is a popular exponentiation in a finite field over integers including prime numbers.
- The integers used by this method are sufficiently large making it difficult to solve.
- There are two sets of keys in this algorithm: private key and public key.



④

RSA alg: Used to generate private key and public key's

Steps:

Here:
$e \rightarrow$ Public key
$d \rightarrow$ Private key

1) choose 2- diff. large no's
2) calculate $n = p * q$
3) calculate $\phi(n) = (p-1) * (q-1)$
4) choose 'e' such that $1 < e < \phi(n)$

(common factor)
e is co-prime to $\phi(n)$, i.e $\gcd(e, \phi(n)) = 1$
(Gcd)

5) Calculate d, such that

[It is a "Euclidean Alg" formula)

$$de \equiv 1 \bmod \phi(n)$$

(Private key generation) (component)

$$\Downarrow \; de = 1 + K \cdot \phi(n)$$

$k = 0, 1, 2, \ldots n$

Encryption:

$$\boxed{\text{cipher } (c) = P^e \bmod n}$$

decryption:

$$\boxed{\text{plaintent } (P) = c^d \bmod n}$$

Public

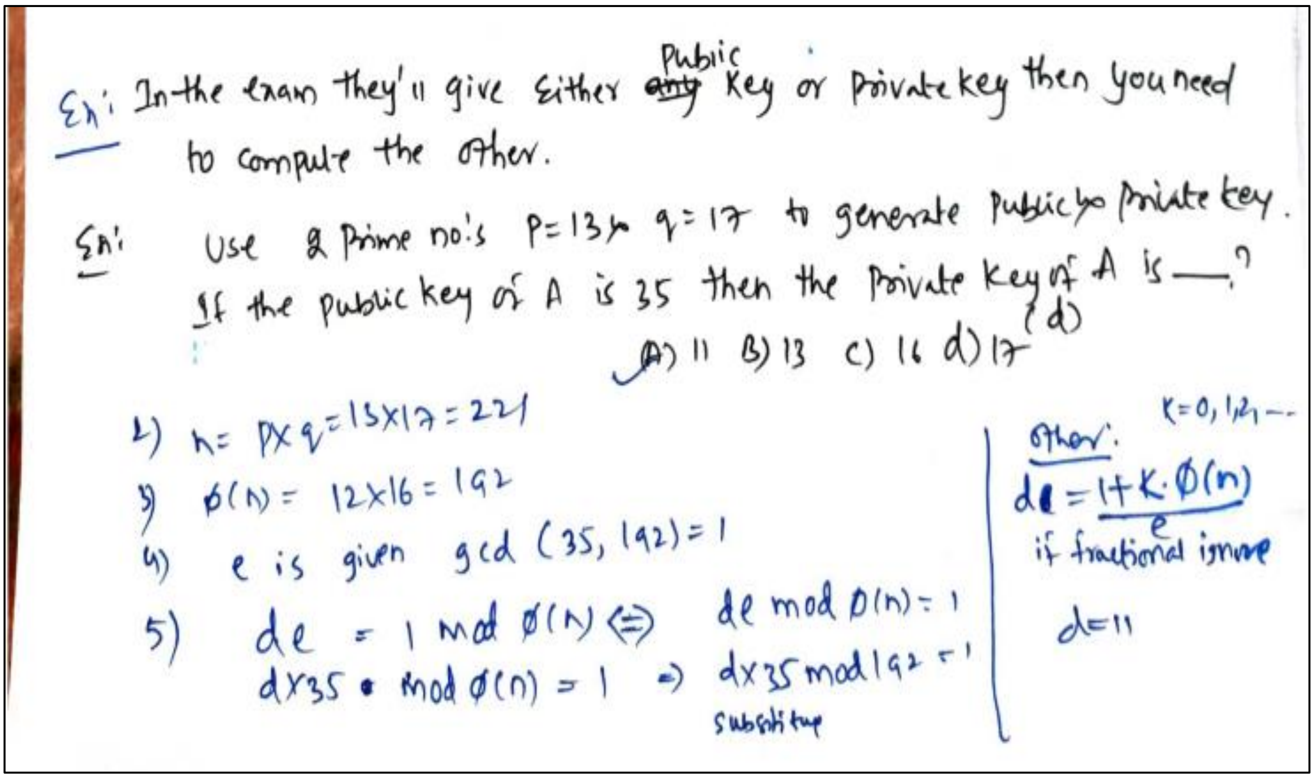
Step 1: Generate the RSA modulus

- The initial procedure begins with selection of two prime numbers namely p and q, and then calculating their product N, as shown −
- N=p*q

  Here, let N be the specified large number.

Step 2: Derived Number (e)
Consider number e as a derived number which should be greater than 1 and less than (p-1) and (q-1). The primary condition will be that there should be no common factor of (p-1) and (q-1) except 1
Step 3: Public key
The specified pair of numbers n and e forms the RSA public key and it is made public.
Step 4: Private Key
Private Key d is calculated from the numbers p, q and e. The mathematical relationship between the numbers is as follows –
ed = 1 mod((p-1)*(q-1))
The above formula is the basic formula for Extended Euclidean Algorithm, which takes p and q as the input parameters.
Encryption Formula
Consider a sender who sends the plain text message to someone whose public key is (n,e). To encrypt the plain text message in the given scenario, use the following syntax:
C = Pe mod n
Decryption Formula
The decryption process is very straightforward and includes analytics for calculation in a systematic approach. Considering receiver C has the private key d, the result modulus will be calculated as −
Plaintext = Cd mod n

Example:



Ex: In the exam they'll give either ~~any~~ Public key or Private key then you need to compute the other.

Ex: Use 2 Prime no's p=13, q=17 to generate Public & Private key. If the public key of A is 35 then the private key of A is ___ ? (d)

[✓ A) 11] B) 13 C) 16 d) 17

1) $n = p \times q = 13 \times 17 = 221$

3) $\phi(n) = 12 \times 16 = 192$

4) e is given $\gcd(35, 192) = 1$

5) $de = 1 \bmod \phi(n) \Leftrightarrow de \bmod \phi(n) = 1$

$d \times 35 \bmod \phi(n) = 1 \Rightarrow \underset{\text{substitute}}{d \times 35 \bmod 192 = 1}$

Other: $k = 0, 1, 2, \ldots$

$d = \frac{1 + k \cdot \phi(n)}{e}$

if fractional ignore

$d = 11$