END TERM EXAMINATION- Sample Paper (1)

B.Tech (ECE)

Subject: Operating System

Time: 3 Hours

Note : Attempt any five questions including Q.no.1 which is compulsory

Q1. Answer the following:-

i. What is the difference between Batch processing, Real time processing, Time sharing and Distributed processing? (2)

Batch processing is assembling a group of jobs or tasks and then doing them all in a batch. Usually something like printing a monthly bill for sending out to customers. Real-time processing is just that, processing calculations right now, in real time. Something like crediting payments to a person's monthly bill would be an example of real time processing. The payments are processed as they come in, on a first come, first served basis. Time sharing is also called time slicing. The CPU divides the number of clock cycles it has per second among several processes. This also goes by the name, multitasking, but that is a misnomer as the CPU only processes one instruction at a time. Think a multilane freeway and cars all merging to cross a river on a 1 lane bridge. Distributed processing means more than one CPU working in parallel. In the freeway analogy, think 1 lane from the freeway splitting and using more than one bridge across the river. The load is distributed over two or more bridges. All of these are just various ways and means of accomplishing one or more tasks from a group of one or more tasks.

ii. Write a short note on "Process Control Block". (2)

- The OS must know specific information about processes in order to manage, control them and also to implement the process model, the OS maintains a table (an array of structures), called the **process table**, with one entry per process.
- These entries are called **process control blocks** (PCB) - also called a task control block.
- Such information is usually grouped into two categories: Process State Information and Process Control Information. Including these:
    - **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.
- **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 OS.
- **Accounting information**: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**: This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

iii. Mention the necessary conditions for a deadlock to occur. (2)

1. ***Mutual Exclusion:*** The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary.
2. ***Hold and wait or partial allocation***: The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.
3. ***No Pre-emption***: The processes must not have resources taken away while that resource is being used. Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.
4. ***Resource waiting or circular wait:*** A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur.

iv. Difference between serial access devices and direct access devices. (2)

With *sequential* access to a storage device, such as with tape, a system enters and retrieves data based on the location of the data, and on a reference to information previously accessed. The closer the physical location of information on the storage device, the quicker the information can be processed.

In contrast, with *direct* access, entering and retrieving information depends only on the location of the data and not on a reference to data previously accessed. Because of this, access time for information on direct access storage devices (DASDs) is effectively independent of the location of the data.

Direct access storage devices (DASDs) include both fixed and removable storage devices. Typically, these devices are hard disks. A *fixed* storage device is any storage device defined during system configuration to be an integral part of the system DASD. If a fixed storage device is not available at some time during normal operation, the operating system detects an error.

A *removable* storage device is any storage device you define during system configuration to be an optional part of the system DASD. Removable storage devices can be removed from the system at any time during normal operation. As long as the device is logically unmounted before you remove it, the operating system does not detect an error.

v. Write a short note on file system mounting. (2)

- The basic idea behind mounting file systems is to combine multiple file systems into one large tree structure.
- The mount command is given a file system to mount and a ***mount point*** (directory) on which to attach it.
- Once a file system is mounted onto a mount point, any further references to that directory actually refer to the root of the mounted file system.
- Any files (or sub-directories) that had been stored in the mount point directory prior to mounting the new file system are now hidden by the mounted file system, and are no longer available. For this reason some systems only allow mounting onto empty directories.
- File systems can only be mounted by root, unless root has previously configured certain files ystems to be mountable onto certain pre-determined mount points. (E.g. root may allow users to mount floppy file systems to /mnt or something like it.) Anyone can run the mount command to see what file systems is currently mounted.

File systems may be mounted read-only, or have other restrictions imposed.

vi. Explain various types of program threats. (2)

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks then it is known as Program Threats. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well known program threats.

Trojan Horse - Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.

Trap Door - If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.

Logic Bomb - Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.

Virus - Virus as name suggest can replicate themselves on computer system .They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user.

vii. Explain the working mechanism of symmetric key and asymmetric key encryption techniques used in cryptography. (3)

Symmetric Encryption:

Symmetric encryption is the oldest and best-known technique. A secret key, which can be a number, a word, or just a string of random letters, is applied to the text of a message to change the content in a particular way. This might be as simple as shifting each letter by a number of places in the alphabet. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.

Asymmetric Encryption:

- The problem with secret keys is exchanging them over the Internet or a large network while preventing them from falling into the wrong hands. Anyone who knows the secret key can decrypt the message. One answer is asymmetric encryption, in which there are two related keys--a key pair. A public key is made freely available to anyone who might want to send you a message. A second, private key is kept secret, so that only you know it.
- Any message (text, binary files, or documents) that are encrypted by using the public key can only be decrypted by applying the same algorithm, but by using the matching private key. Any message that is encrypted by using the private key can only be decrypted by using the matching public key.
- This means that you do not have to worry about passing public keys over the Internet (the keys are supposed to be public). A problem with asymmetric encryption, however, is that it is slower than symmetric encryption. It requires far more processing power to both encrypt and decrypt the content of the message.

UNIT - I

Q2 (a). Explain the different types of operating system. (5)

There are 4 types of operating system which is explained below:-

i. Batch Operating System:-A batch system is one in which jobs are bundled together with the instructions necessary to allow them to be processed without intervention.

In a multitasking computer system, processes may occupy a variety of states. These distinct states may not actually be recognized as such by the operating system kernel, however they are a useful abstraction for the understanding of processes. The monitor is system software that is responsible for interpreting and carrying out the instructions in the batch jobs. When the monitor starts a job, the entire computer is dedicated to the job, which then controls the computer until it finishes.

ii. Multiprogramming Operating System:- As machines with more and more memory became available, it was possible to extend the idea of multiprogramming (or multiprocessing) as used in spooling batch systems to create systems that would load several jobs into memory at once and cycle through them in some order, working on each one for a specified period of time. There is different type of Multiprogramming Operating System, some main stream are discussed below:-

a. Multitasking Operating System:- A running state of a program is called a process or a task. Multitasking allows the computer system to more reliably guarantee each process a regular "slice" of operating time. It also allows the system to rapidly deal with important external events like incoming data, which might require the immediate attention of one or another process. So, multitasking operating system is a type of multiprogramming operating system which can perform several process simultaneously. The earliest multitasking OS available to home users was the AmigaOS. All current major operating system support this feature.

b. Multi-user Operating System:- A multi-user operating system allows for multiple users to use the same computer at the same time and/or different times. Linux,Unix,Windows OS are some example of multitasking operating system.

c. Multiprocessing Operating System:- An operating system capable of supporting and utilizing more than one computer processor.Linux,Unix,Windows OS are some example of multitasking operating system.

d. Real Time Operating System:- It is an OS where there are a number of possibly unrelated external activities needed to be controlled by a single processor system. In such systems a hierarchical interrupt system was coupled with process prioritization to ensure that key activities were given a greater share of available process time.

iii. Network Operating System:- A network operating system (NOS) is software that controls a network and its message (e.g. packet) traffic and queues, controls access by multiple users to network resources such as files, and provides for certain administrative functions, including security.

iv. Distributed Operating System:- Distributed systems are very much like traditional operating systems. First, they act as resource managers for the underlying hardware, allowing multiple users and applications to share resources such as CPUs, memories, peripheral devices, the network, and data of all kinds. Second, and perhaps more important, is that distributed

systems attempt to hide the intricacies and heterogeneous nature of the underlying hardware by providing a virtual machine on which applications can be easily executed.

Q2(b). Explain Process Control Block. Draw the block diagram of process transition states. (5)

Process Control block is used for storing the collection of information about the Processes and this is also called as the Data Structure which Stores the information about the process. The information of the Process is used by the CPU at the Run time. The various information which is Stored into the PCB as followings:

1) Name of the Process.

2) State of the Process. Means Ready, Active, Wait.

3) Resources allocated to the Process

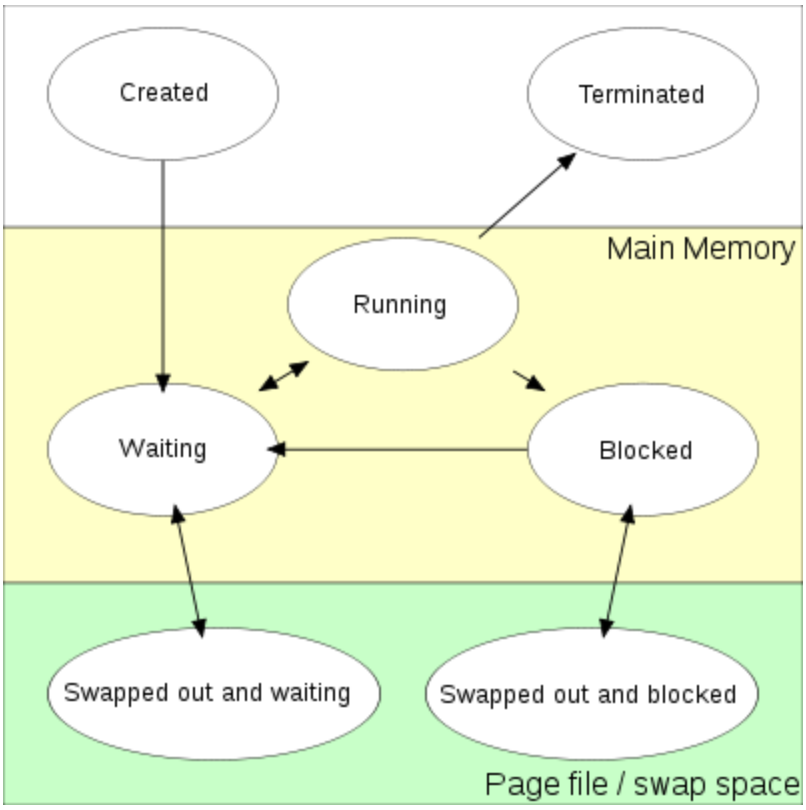
4) Memory which is provided to the Process.

5) Scheduling information.

6) Input and Output Devices used by the Process.

7) Process ID or a Identification Number which is given by the CPU when a Process Request for a Service.

Process states:



Q 2(c). Difference between process and thread. ***(5)***

Process:
Each process provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads.

Thread:
A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

Another difference between a thread and a process is that threads within the same process share the same address space, whereas different processes do not. This allows threads to read from and write to the same data structures and variables, and also facilitates communication between

threads. Communication between processes – also known as IPC, or inter-process communication – is quite difficult and resource-intensive.

OR

Q3(a). What do you mean by system call. Discuss system call parameters. Also discuss discuss different types of system calls. (10)

System calls provide an interface between the process an the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed. For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). In some sense, they are bundles of useful system calls.

The system call provides an interface to the operating system services.

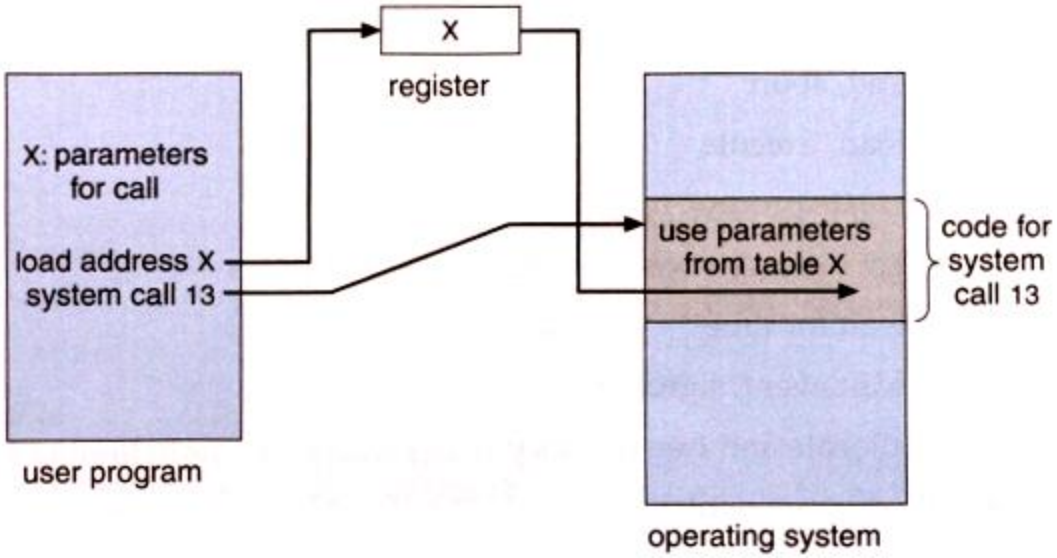
Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained:

- Portability: as long a system supports an API, any program using that API can compile and run.
- Ease of Use: using the API can be significantly easier then using the actual system call.

System Call Parameters:

Three general methods exist for passing parameters to the OS:

1. Parameters can be passed in registers.
2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
3. Parameters can also be pushed on or popped off the stack by the operating system.



Types of System Calls:

There are 5 different categories of system calls:

process control, file manipulation, device manipulation, information maintenance and communication.

Process Control:

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

File Management:

Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*. Also, there is a need to determine the file attributes – *get* and *set* file attribute. Many times the OS provides an API to make these system calls.

Device Management:

Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs *request* the device, and when finished they *release* the device. Similar to files, we can *read*, *write*, and *reposition* the device.

Information Management:

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is *time*, or *date*.

The OS also keeps information about all its processes and provides system calls to report this information.

Communication:

There are two models of interprocess communication, the message-passing model and the shared memory model.

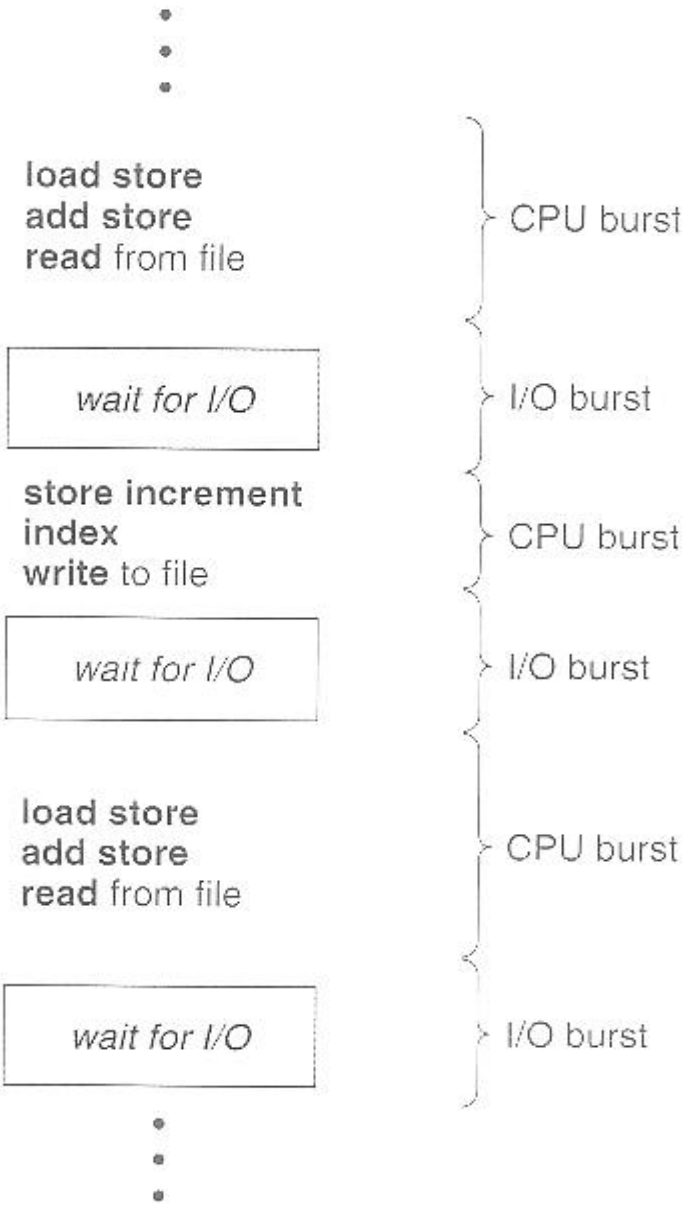
- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

Q3(b). What do you mean by CPU scheduling? Discuss CPU/IO burst cycle. (5)

- Almost all programs have some alternating cycle of CPU number crunching and waiting for I/O of some kind. (Even a simple fetch from memory takes a long time relative to CPU speeds.)
- In a simple system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever.
- A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of otherwise lost CPU cycles.
- The challenge is to make the overall system as "efficient" and "fair" as possible, subject to varying and often dynamic conditions, and where "efficient" and "fair" are somewhat subjective terms, often subject to shifting priority policies.

CPU-I/O Burst Cycle:

- Almost all processes alternate between two states in a continuing ***cycle***, as shown in below figure***:***
    - A CPU burst of performing calculations, and
    - An I/O burst, waiting for data transfer in or out of the system.



UNIT-2

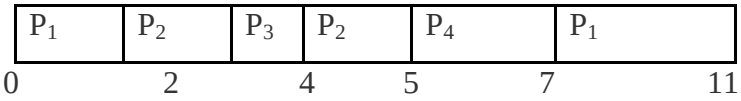
Q4(a). Consider the following process: **(10)**

PROCESS	ARRIVAL TIME	SERVICE TIME
P1	0	7
P2	2	4
P3	4	1

P4	5	4

Solve the above problem with Shortest Remaining Time First by drawing Gantt chart and also calculate the average waiting time and turnaround time. Also calculate throughput.

Gantt chart



PROCESS	WAIT TIME	TURN AROUND TIME
P1	11 – 2 = 9	16 – 0 = 16
P2	(2 – 2) + (11 – 4) = 7	7 – 2 = 5
P3	4 – 4 = 0	5 – 4 = 1
P4	7 – 5 = 2	11 – 5 = 6

Total Wait Time
9 + 7 + 0 + 2 = 18 ms

Average Waiting Time = (Total Wait Time) / (Total number of processes)
18/4 = 4.5 ms

Total Turn Around Time
16 + 5 + 1 + 6 = 28 ms

Average Turn Around time = (Total Turn Around Time) / (Total number of processes)
28/4 = 7 ms

Throughput
4 jobs/16 sec = 0.25 jobs/sec

Q4(b). Discuss differences between paging and segmentation. (5)

- ***Paging:*** Computer memory is divided into small partitions that are all the same size and referred to as, page frames. Then when a process is loaded it gets divided into pages which are the same size as those previous frames. The process pages are then loaded into the frames.

  Segmentation: Computer memory is allocated in various sizes (segments) depending on

the need for address space by the process. These segments may be individually protected or shared between processes. Commonly you will see what are called "Segmentation Faults" in programs, this is because the data that's is about to be read or written is outside the permitted address space of that process.

So now we can distinguish the differences and look at a comparison between the two:

Paging:
Transparent to programmer (system allocates memory)
No separate protection
No separate compiling
No shared code

Segmentation:
Involves programmer (allocates memory to specific function inside code)
Separate compiling
Separate protection
Share code

OR

Q5. Discuss the dining philosopher problem with its solution and reader writer problem. (15)

The Dining-Philosophers Problem:

- The dining philosopher's problem is useful for modeling processes that are competing for exclusive access to a limited number of resources, such as I/O devices.
- Consider five philosophers who spend their lives thinking and eating.
- The philosophers share a 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 dining-philosophers problem 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-free and starvation-free manner.
- One simple solution is to represent each chopstick with a semaphore.
- A philosopher tries to grab a 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.

Solution of Dining-Philosophers Problem:

- Although this solution guarantees that no two neighbors are eating simultaneously, it nevertheless must be rejected because it could create 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.
- One improvement is that has no deadlock and no starvation is to protect the five statements following the call to think by a binary semaphore.

- Before starting to acquire forks, a philosopher would do a down on mutex
- After replacing the forks, she would do an up on mutex
- It has a performance bug: only one philosopher can be eating at any instant. With five forks available, we should be able to allow two philosophers to eat at the same time.
- Any satisfactory solution to the dining-philosophers problem must guard against the possibility that one of the philosophers will starve to death. A deadlock-free solution does not necessarily eliminate the possibility of starvation.

Reader-writer problem:

The readers and writers problem is quite straightforward: we have a database which many users are allowed to read simultaneously, but to which only one user may write at a time (to avoid race conditions in the database). So it is identical to the mutual exclusion problem, except that we wish to permit more flexibility in the system by letting more than one simultaneous reader. Naturally there is no danger of a race condition when many tasks are *reading* from a shared data structure.

Solution of reader-writer problem:

We can solve this problem using---guess what---semaphores. The approach is to have one mutual exclusion semaphore for the database as a whole. Essentially though, all the readers share the same critical section, that is, only the first reader does a WAIT on the database mutex

semaphore. All subsequent readers notice that another reader has locked the database, and proceed to read. When the last reader has finished, it 'leaves' the critical section by performing a SIGNAL on the semaphore. It's like: last reader out switches off the light.

Any writer process must protect its critical section by Waiting and Signaling as normal (since it may not share the database with readers, nor with other writers).

Since the readers must keep track of how many of them there are, they share an integer counter variable (called 'numberOfReaders'), which must also be protected by a semaphore.

So the variables shared by all readers and writers are...

```
Semaphore databaseMutex = new Semaphore(1);    // Binary semaphores again.
Semaphore readersMutex = new Semaphore(1);
integer numberOfReaders = 0;
```

The code for a writer is pretty simple, so let's look at that first:

```
void write(some stuff){
   databaseMutex.wait();
   // Write some stuff to the database.
   databaseMutex.signal();
}
```

As you can see, it's standard mutual exclusion. The code for the readers, however, must only lock the 'databaseMutex' if it is the first reader, and must only unlock the 'databaseMutex' if it is the last reader (allowing for other readers to come and go while we are reading).

```
resulttype read(some thing){
   readersMutex.wait();
   numberOfReaders += 1;
   if (numberOfReaders == 1) databaseMutex.wait();
   readersMutex.signal();

   // Read something from database.

   readersMutex.wait();
   if (numberOfReaders == 1) databaseMutex.signal();
   numberOfReaders -= 1;
   readersMutex.signal();
   return result;
}
```

We must check when leaving the reading routine whether we were the last to leave. It does *not* stand to reason that the same reader that locked the database is the one to unlock it. We may be the first reader, but other readers join while we are reading, and they are still reading after we leave. Someone else can take the responsibility of unlocking the database again.

UNIT-III

Q 6(a). Differentiate between blocking vs. non-blocking I/O. (5)

Some control over how the wait for I/O to complete is accommodated is available to the programmer of user applications. Most I/O requests are considered **blocking** requests, meaning that control does not return to the application until the I/O is complete. The delayed from systems calls such read () and write () can quite long. Using systems call that block is sometimes call **synchronous** programming. In most cases, the wait is not really a problem because the program cannot do anything else until the I/O is finished. However, in cases such as network programming with multiple clients or with graphical user interface programming, the program wish to perform other activity as it continues to wait for more data or input from users.

One solution for these situations is to use multiple threads so that one part of the program is not waiting for unrelated I/O to complete. Another alternative is to use **asynchronous** programming techniques with **nonblocking** system calls. An asynchronous call returns immediately, without waiting for the I/O to complete. The completion of the I/O is later communicated to the application either through the setting of some variable in the application or through the triggering of a signal or call-back routine that is executed outside the linear control flow of the application.

Q6(b). Suppose that a disk drive has 200 cylinders, numbered 0 to 199.The work queue is: 23, 89, 132, 42, 187. (10)
Determine the total distance for the following disk scheduling algorithms:

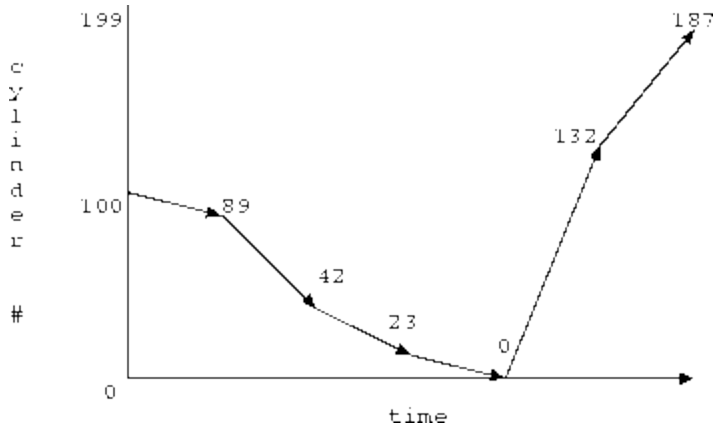
(i) SCAN
(ii) LOOK
(iii) C-SCAN
(iv) C-LOOK

Work Queue**: 23, 89, 132, 42, 187**

- there are 200 cylinders numbered from 0 - 199
- the diskhead stars at number 100

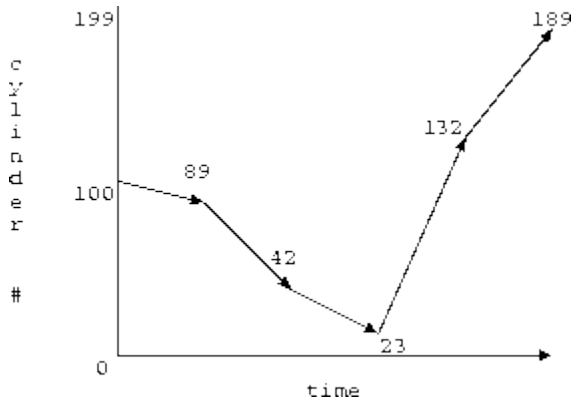
- **SCAN**

- assume we are going inwards (i.e., towards 0)



$$|100-89|+|89-42|+|42-23|+|23-0|+|0-132|+|132-187|=11+47+19+23+1$$

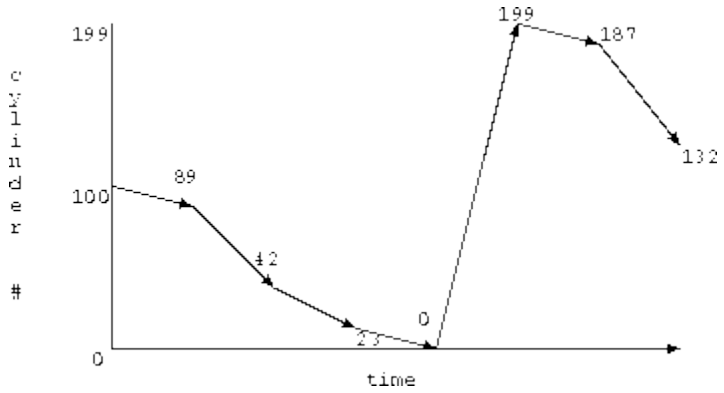
- **LOOK**



$$|100-89|+|89-42|+|42-23|+|23-132|+|132-187|=11+47+19+109+55=241$$

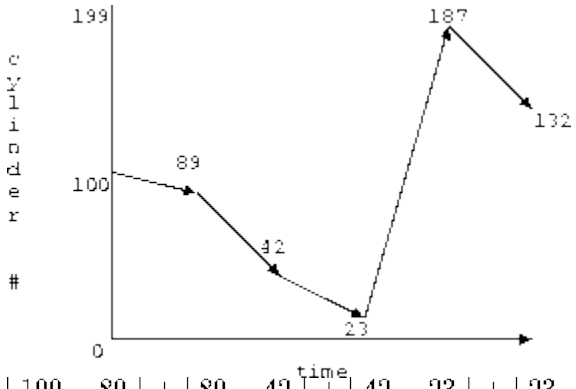
- reduce variance compared to SCAN

- **C-SCAN**



$$|100-89|+|89-42|+|42-23|+|23-0|+|0-199|+|199-187|+|187-132|=11+4$$

- **C-LOOK**



$$|100-89|+|89-42|+|42-23|+|23-187|+|187-132|=11+47+19+164+55=296$$

OR

Q7(a). What do you mean by RAID Structure? Also discuss different types of RAID levels. (10)

RAID stands for redundant array of independent disks. The name indicates that the disk drives are independent, and are multiple in number. How the data is distributed between these drives depends on the RAID level used.

The main advantage of RAID, is the fact that, to the operating system the array of disks can be presented as a single disk.

RAID is fault tolerant because in most of the RAID level's data is redundant in multiple disks, so even if one disk fails, or even two sometimes, the data will be safe and the operating system will not be even aware of the failure. DATA loss is prevented due to the fact that data can be recovered from the disk that are not failed.

RAID levels:

RAID 0:

RAID 0 includes a disk array that implements striping without any drive redundancy. It offers no fault tolerance and is less reliable than a single-drive implementation; its only advantage is speed. RAID 0 is suitable for certain special applications, as in scientific analysis or imaging,

where compromised system reliability can be tolerated.

RAID :

RAID 1 is disk mirroring. Two drives store identical information so that one is a mirror of the other. For every disk operation, the system must write the same information to both disks. Because dual write operations can degrade system performance, many employ duplexing, where each mirror drive has it own host adapter. While the mirror approach provides good fault tolerance, it is relatively expensive to implement, because only half of the available disk space can be used for storage while the other is used for mirroring. Novell NetWare, in particular, incorporates support for disk mirroring.

RAID 2:

RAID 2 uses extra check disks, with data bits striped across the data and check disks. The data includes an interleaved Hamming code, which can be used to detect and correct single bit errors as well as detect double bit errors. Because of the amount of information required for the check bits, several check disks are required to implement RAID 2. It is optimal for reading and writing large data blocks at high data transfer rates, but smaller block reads are inefficient. Read, modify, and write operations required for small block write operation also result in poor performance. RAID 2 is generally impractical for smaller systems and is not available with Microsoft Windows NT Advanced Server.

RAID 3:

RAID 3 uses a single redundant check disk (sometimes referred to as a parity disk) for each group of drives. Data written to the RAID 3 disk array is bit striped across the data disks. The check disk receives the XOR (exclusive OR) of all the data values written to the data drives. Because data transfers to and from individual drives occur only in unit sector multiples, the minimum amount of data that can be written to or read from RAID 3 disk array is the number of data drives multiplied by the number of bytes per sector (this is known as a transfer unit). This option is not available with Microsoft Windows NT Advanced Server.

RAID 4:

RAID 4 offers a disk array architecture that is better optimized for transaction processing applications than RAID 3. RAID 4 performs block striping or sector striping on the data on the drives, while RAID 3 performs bit striping. Thus, with RAID 4, one entire sector is written to one drive, the next sector is written to the next drive, and so on. This technique allows multiple

unrelated sectors to be read simultaneously, and it is particularly valuable for small reads that need to access only a single drive in the array. RAID 4 dedicates one entire disk for storing check data, allowing data from a failed drive to be easily recovered. While this approach allows multiple reads to occur simultaneously, with different sectors from different drives, write operations are bottlenecked. Because the single check disk operation must be written to during every write operation, only one write operation can take place at a time. This option is not available with Microsoft Windows NT Advanced Server.

RAID 5:

Unlike RAID 4, which dedicates a single physical disk for check data, RAID 5 dedicates the equivalent of one entire disk for storing check data but distributes the check data over all the drives in the group. For example, sector 1 of disk 5 may be assigned to hold the check data for sector 1 of the remaining data drives and so on. Because the check data is simply the XOR of all the write data values for the corresponding sector on each of the data disks, as long as the old sector data and the old check data values are known, the new check data for a single sector write can be calculated without having to read the corresponding sectors from the other data disks. Thus, only two disks are involved in a single sector write operation: the target data disk and the corresponding disk that holds the check data for that sector. This is in contrast to the RAID 3 implementation, which requires all drives in a group to be read and written when a single sector size write occurs. The primary benefit of the RAID 5 distributed check data approach is that it permits write operations to take place simultaneously. It also allows multiple reads to take place simultaneously and is efficient in handling small amounts of information. This is the preferred option when setting up fault tolerance in Microsoft Windows NT Advanced Server.

Q7(b). Write a short note on disk scheduling algorithm. (5)

In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk Scheduling Algorithms are used to reduce the total seek time of any request. The purpose of this material is to provide one with help on disk scheduling algorithms. Hopefully with this, one will be able to get a stronger grasp of what disk scheduling algorithms do.

Different types of disk scheduling algorithms:

Although there are other algorithms that reduce the seek time of all requests, there are the following disk scheduling algorithms:
First Come-First Serve (FCFS)
Shortest Seek Time First (SSTF)
Elevator (SCAN)
Circular SCAN (C-SCAN)

LOOK
C-LOOK

These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be.

UNIT- IV

Q8. What do you mean by directory structure? Also discuss different types of directory structures. (15)

Storage *Structure:*

- A disk can be used in its entirety for a file system.
- Alternatively a physical disk can be broken up into multiple ***partitions, slices, or mini-disks***, each of which become a virtual disk and can have its own file system. (or be used for raw storage, swap space, etc.)
- Or, multiple physical disks can be combined into one ***volume***, i.e. a larger virtual disk, with its own file system spanning the physical disks.

Directory:

- Directory operations to be supported include:
  - Search for a file
  - Create a file - add to the directory
  - Delete a file - erase from the directory
  - List a directory - possibly ordered in different ways.
  - Rename a file - may change sorting order
  - Traverse the file system.

Single-Level Directory:

- Simple to implement, but each file must have a unique name.

- ***Limitations of Single Level Directory***

  a) Since all files are in the same directory, they must have unique name.

  b) If two user call their data free test, then the unique name rule is violated.

c) Files are limited in length.

d) Even a single user may find it difficult to remember the names of all files as the number of file increases.

e) Keeping track of so many file is daunting task.

Two-Level *Directory:*

- Each user gets their own directory space.
- File names only need to be unique within a given user's directory.
- A master file directory is used to keep track of each users directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system (executable) files.
- Systems may or may not allow users to access other directories besides their own
    - If access to other directories is allowed, then provision must be made to specify the directory being accessed.
    - If access is denied, then special consideration must be made for users to run programs located in system directories. A ***search path*** is the list of directories in which to search for executable programs, and can be set uniquely for each user.
        - Each user has Its own User File Directory (UFD).
        - When the user job start or user log in, the system Master File Directory (MFD) is searched. MFD is indexed by user name or Account Number.
        - When user refers to a particular file, only his own UFD is searched.

          Thus different users may have files with same name. To have a particular file uniquely, in a two level directory, we must give both the user name and file name. *A two level directory can be a tree or an inverted tree of height 2.* The root of a tree is Master File Directory (MFD). Its direct descendents are User File Directory (UFD). The descendents of UFD's are file themselves. The files are the leaves of the tree.

        - ***Limitations of Two Level Directory:*** The structure effectively isolates one user from another.

Tree-Structured Directories:

- Each user / process has the concept of a ***current directory*** from which all (relative) searches take place.
- Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory)

- Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.
- One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.
- Deletions if directory is empty, its entry in the directory that contains it can simply deleted. If it is not empty : One of the Two approaches can be taken :-
  a)User must delete all the files in the directory.

  b)If any sub directories exist, same procedure must be applied.
  The **UNIX rm command** is used.
  MS dos will not delete a directory unless it is empty.

Acyclic-Graph Directories:

- When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user / process), it can be useful to provide an acyclic-graph structure.
  - UNIX provides two types of ***links*** for implementing the acyclic-graph structure.
  - A ***hard link*** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same file system.
  - A ***symbolic link***, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other file systems, as well as ordinary files in the current file system.
- Windows only supports symbolic links, termed ***shortcuts.***
- Hard links require a ***reference count***, or ***link count*** for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.
- For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted:
  - One option is to find all the symbolic links and adjust them also.
  - Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used.
  - What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?
  - Acyclic Graph is the graph with no cycles. It allows directories to share sub directories and files. With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the another.

General Graph Directory:

- If cycles are allowed in the graphs, then several problems can arise:

- Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories.)
- Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem. (chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system. Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted.)

OR

Q9. Write short notes on the following:- (5X3=15)

(a). Intrusion detection

Intrusion detection (ID) is a type of security management system for computers and networks. An ID system gathers and analyzes information from various areas within a computer or a network to identify possible security breaches, which include both intrusions (attacks from outside the organization) and misuse (attacks from within the organization). ID uses *vulnerability assessment* (sometimes referred to as *scanning*), which is a technology developed to assess the security of a computer system or network. An intrusion detection system (IDS) monitors network traffic and monitors for suspicious activity and alerts the system or network administrator. In some cases the IDS may also respond to anomalous or malicious traffic by taking action such as blocking the user or source IP address from accessing the network.

IDS come in a variety of "flavors" and approach the goal of detecting suspicious traffic in different ways. There are network based (NIDS) and host based (HIDS) intrusion detection systems. There are IDS that detect based on looking for specific signatures of known threats- similar to the way antivirus software typically detects and protects against malware- and there are IDS that detect based on comparing traffic patterns against a baseline and looking for anomalies.

Intrusion detection functions include:

- Monitoring and analyzing both user and system activities
- Analyzing system configurations and vulnerabilities
- Assessing system and file integrity
- Ability to recognize patterns typical of attacks
- Analysis of abnormal activity patterns
- Tracking user policy violations

(b). System Threats

System Threats

System threats refer to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. A system threat creates such an environment that operating system resources/ user files are misused. Following is the list of some well known system threats.

- **Worm** -Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.
- **Port Scanning** - Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service** - Denial of service attacks normally prevents user to make legitimate use of the system. For example user may not be able to use internet if denial of service attacks browser's content settings.

(c). User Authentication and the concept of "one-time password"

Authentication:

Authentication refers to identifying the each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways:

- **Username / Password** - User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** - User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** - User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One time passwords provides additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used then it cannot be used again. One time password are implemented in various ways.

- **Random numbers** - Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.
- **Secret key** - User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.

- **Network password** - Some commercial applications send one time password to user on registered mobile/ email which is required to be entered prior to login.