OS * A Program that control the execution of application program
* An Interface between application & Hardware.
* Controls & coordinates usage of hardware among various applicat & user * main objective of an OS : - convenient, - efficiency, - ability to evolve

OS Operation & function (what does OS do?)

* Security: - OS uses password protection to protect user data
 - prevent unauthorised access to program & data.
* Job accounting: - OS keeps tracks of time & resources by various tasks
 - It can be used to track usage resource for a particular group
* Error detecting aids: - OS constantly monitors the system to detect errors and avoid the malfunctioning of computer.
* Memory management: - The OS manages the Primary memory and main memory. - It keeps the tracks of primary memory.
 - In multiprogramming, OS decides the order in which process have access to memory. * Process management: - In multiprogramming environment, OS decides the order in which processes have access the processor. - The function of OS is called process scheduling.
* Device management: - OS manage device communication via drivers.
 - Decides which process gets access to certain devices & for how long
* File management: - OS keeps tracks of where information is stored, user access & status of every file. - These facilities are known as file system.

OS SERVICES

Program Execution:

- The Operating System is responsible for execution of all types of programs whether it be user programs or system programs.
- The Operating System utilises various resources available for the efficient running of all types of functionalities.

Handling Input/Output Operations:

- The Operating System is responsible for handling all sorts of inputs, i.e, from keyboard, mouse, desktop, etc.
- The Operating System does all interfacing in the most appropriate manner regarding all kinds of Inputs and Outputs.

Manipulation of File System:

- The Operating System is responsible for making decisions regarding the storage of all types of data or files, i.e, floppy disk/hard disk/pen drive, etc.
- The Operating System decides how the data should be manipulated and stored.

Error Detection and Handling:

- The Operating System is responsible for detection of any types of error or bugs that can occur while any task.
- The well secured OS sometimes also acts as countermeasure for preventing any sort of breach to the Computer System from any external source and probably handling them.

Resource Allocation:

- The Operating System ensures the proper use of all the resources available by deciding which resource to be used by whom for how much time.
- All the decisions are taken by the Operating System.

Accounting:

- The Operating System tracks an account of all the functionalities taking place in the computer system at a time.
- All the details such as the types of errors occurred are recorded by the Operating System.

Information and Resource Protection:

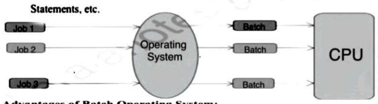
- The Operating System is responsible for using all the information and resources available on the machine in the most protected way.
- The Operating System must foil an attempt from any external resource to hamper any sort of data or information.

Module1(1)

Types of Operating Systems:

1. Batch Operating System –

- This type of operating system does not interact with the computer directly.
- There is an operator which takes similar jobs having the same requirement and groups them into batches.
- It is the responsibility of the operator to sort jobs with similar needs.
- Examples of Batch based Operating System: Payroll System, Bank Statements, etc.



Advantages of Batch Operating System:

- It is very difficult to guess or know the time required for any job to complete. Processors of the batch systems know how long the job would be when it is in queue
- Multiple users can share the batch systems
- The idle time for the batch system is very less
- It is easy to manage large work repeatedly in batch systems

Disadvantages of Batch Operating System:

- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometimes costly
- The other jobs will have to wait for an unknown time if any job fails

2. Time-Sharing Operating Systems –

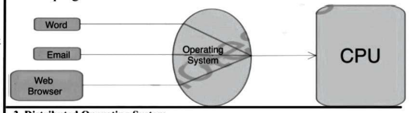
- Each task is given some time to execute so that all the tasks work smoothly.
- Each user gets the time of CPU as they use a single system.
- These systems are also known as Multitasking Systems.
- The task can be from a single user or different users also.

Advantages of Time-Sharing OS:

- Each task gets an equal opportunity
- Fewer chances of duplication of software
- CPU idle time can be reduced

Disadvantages of Time-Sharing OS:

- Reliability problem
- One must have to take care of the security and integrity of user programs and data



3. Distributed Operating System –

- These types of operating systems are a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, with a great pace.
- Various autonomous interconnected computers communicate with each other using a shared communication network.
- Independent systems possess their own memory unit and CPU.
- These are referred to as loosely coupled systems or distributed systems.

vantages of Distributed Operating System:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed

sadvantages of Distributed Operating System:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which is used are not well defined yet

Real-Time Operating System –

- These types of OSs serve real-time systems.
- The time interval required to process and respond to inputs is very small.
- This time interval is called response time.

Module1(2)

Two types of Real-Time Operating System which are as follows:

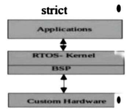
- Hard Real-Time Systems:
  - These OSs are meant for applications where time constraints are very strict and even the shortest possible delay is not

Hard Real-Time Systems:

- These OSs are meant for applications where time constraints are very strict and even the shortest possible delay is not acceptable.
- These systems are built for saving life like automatic parachutes or airbags which are required to be readily available in case of any accident.
- Virtual memory is rarely found in these systems.

Soft Real-Time Systems:

- These OSs are for applications where time-constraint is less strict



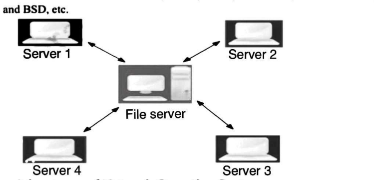
- Examples of Real-Time Operating Systems are: Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.
- Advantages of RTOS:
  - Memory Allocation: Memory allocation is best managed in these types of systems.

Disadvantages of RTOS:

- Limited Tasks: Very few tasks run at the same time and their concentration is very less on few applications to avoid errors.
- Use heavy system resources: Sometimes the system resources are not so good and they are expensive as well.

Network Operating System –

- These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions.
- These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network.
- One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections, etc. and that's why these computers are popularly known as tightly coupled systems.
- Examples of Network Operating System are: Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD, etc.



Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on a central location for most operations
- Maintenance and updates are required regularly

Module1(3)

Multiprocessing system

Asymmetric	Symmetric
In asymmetric multiprocessing, the processors are not treated equally.	In symmetric multiprocessing, all the processors are treated equally.
Tasks of the operating system are done by the master processor.	Tasks of the operating system are done individual processor
No Communication between Processors as they are controlled by the master processor.	All processors communicate with another processor by a shared memory.
In asymmetric multiprocessing, processes are master-slave.	In symmetric multiprocessing, the process is taken from the ready queue.
Asymmetric multiprocessing systems are cheaper.	Symmetric multiprocessing systems are costlier.
Asymmetric multiprocessing systems are easier to design	Symmetric multiprocessing systems are complex to design

Advantages of Multiprocessor Systems

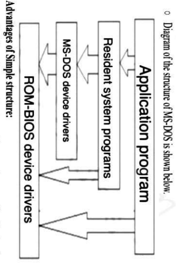
- **More reliable Systems**
  - In a multiprocessor system, even if one processor fails, the system will not halt.
  - This ability to continue working despite hardware failure is known as graceful degradation.
- **Enhanced Throughput**
  - If multiple processors are working in tandem, then the throughput of the system increases i.e. number of processes getting executed per unit of time increase.
  - If there are N processors then the throughput increases by an amount just under N.
- **More Economic Systems**
  - Multiprocessor systems are cheaper than single processor systems in the long run because they share the data storage, peripheral devices, power supplies etc.
  - If there are multiple processes that share data, it is better to schedule them on multiprocessor systems with shared data than have different computer systems with multiple copies of the data.

Disadvantages of Multiprocessor Systems

- **Increased Expense**
  - Even though multiprocessor systems are cheaper in the long run than using multiple computer systems, still they are quite expensive.
  - It is much cheaper to buy a simple single processor system than a multiprocessor system.
- **Complicated Operating System Required**
  - There are multiple processors in a multiprocessor system that share peripherals, memory etc.
  - So, it is much more complicated to schedule processes and impart resources to processes than in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.
- **Large Main Memory Required**
  - All the processors in the multiprocessor system share the memory. So a much larger pool of memory is required as compared to single processor systems.

Simple structure:

- Such operating systems do not have well defined structure and are small, simple and limited systems.
- The interfaces and levels of functionality are not well separated.
- MS-DOS is an example of such an operating system.
- In MS-DOS application programs are able to access the basic I/O routines.
- These types of operating systems cause the entire system to crash if one of the user programs fails.
- Diagram of the structure of MS-DOS is shown below.



Advantages of Simple structure:

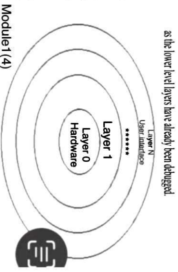
- It delivers better application performance because of the few interfaces between the application program and the hardware.
- Easy for kernel developers to develop such an operating system.

Disadvantages of Simple structure:

- The structure is very complicated as no clear boundaries exist between modules.
- It does not enforce data hiding in the operating system.

Layered structure:

- An OS can be broken into pieces and retain much more control on the system.
- In this structure the OS is broken into a number of layers (levels).
- The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface.
- These layers are so designed that each layer uses the functions of the lower level layers only.
- This simplifies the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged.



Module1(4)

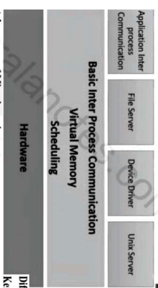
Advantages of Layered structure:

- Layering makes it easier to enhance the operating system as implementation of a layer can be changed easily without affecting the other layers.
- It is very easy to perform debugging and system verification.

Disadvantages of Layered structure:

- In this structure the application performance is degraded as compared to simple structure.
- It requires careful planning for designing the layers as higher layers use the functionalities of only the lower layers.

- This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs.
- This results in a smaller kernel called the micro-kernel.
- Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified.
- Mac OS is an example of this type of OS.
- Microkernel architecture is small and isolated therefore it can function better.
- Providing services in a microkernel system are expensive compared to the normal monolithic system



Advantages of Microkernel structure:

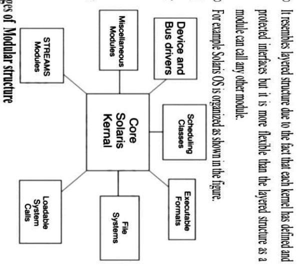
- It makes the operating system portable to various platforms.
- As microkernels are small so these can be tested effectively.

Disadvantages of Microkernel structure:

- Increased level of inter module communication degrades system performance.

Modular structure or approach:

- It is considered as the best approach for an OS.
- It involves designing a modular kernel.
- The kernel has only a set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time.
- It resembles layered structure due to the fact that each kernel has defined and protected interfaces but it is more flexible than the layered structure as a module can call any other module.
- For example Solaris OS is organized as shown in the figure.



Advantages of Modular structure

- advantages of layered structure but with more flexible
- advantages of microkernel approach, without message passing overhead

Disadvantages of Modular structure

- not as clean a design as the layered approach
- not as small a kernel as a microkernel
- but, achieves best of both worlds as far as possible

Differences Between Microkernel and Monolithic Kernel

- The microkernel is much smaller in size as compared to the monolithic kernel.
- The microkernel is easily extensible whereas this is quite complicated for the monolithic kernel.
- The execution of the microkernel is slower as compared to the monolithic kernel.
- Much more code is required to write a microkernel than the monolithic kernel.
- Examples of Microkernel are QNX, Symbian, L4 Linux etc. Monolithic Kernel examples areLinux, BSD etc.

Module1(5)

Types of system calls

Process Control

- This system calls perform the task of process creation, process termination, etc.
- Functions:
  - End and Abort
  - Load and Execute
  - Create Process and Terminate Process
  - Wait and Signed Event
  - Allocate and free memory

File Management

- File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.
- Functions:
  - Create a file
  - Delete file
  - Open and close file
  - Read, write, and reposition
  - Get and set file attributes

Device Management

- Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
- Functions
  - Request and release device
  - Logically attach/ detach devices
  - Get and Set device attributes

Information Maintenance

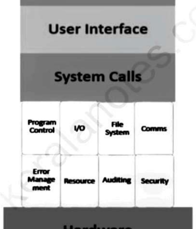
- It handles information and its transfer between the OS and the user program.
- Functions:
  - Get or set time and date
  - Get process and device attributes

Communications

- These types of system calls are specially used for interprocess communications.
- Functions:
  - Create, delete communications connections
  - Send, receive message
  - Help OS to transfer status information
  - Attach or detach remote devices

System calls

- A system call is a mechanism that provides the interface between a process and the operating system.
- It is a programmatic method in which a computer program requests a service from the kernel of the OS.
- System call offers the services of the operating system to the user programs via API (Application Programming Interface).
- System calls are the only entry points for the kernel system.



Why do you need System Calls in OS?

- Reading and writing from files demand system calls.
- If a file system wants to create or delete files, system calls are required.
- System calls are used for the creation and management of new processes.
- Network connections need system calls for sending and receiving packets
- Access to hardware devices like scanner, printer, need a system call.

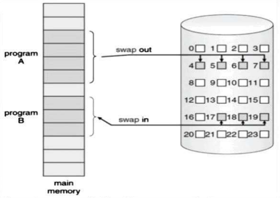
How Does System Call Work?

- **Step 1)** The processes executed in the user mode till the time a system call interrupts it.
- **Step 2)** After that, the system call is executed in the kernel-mode on a priority basis.
- **Step 3)** Once system call execution is over, control returns to the user mode.,
- **Step 4)** The execution of user processes resumed in Kernel mode.

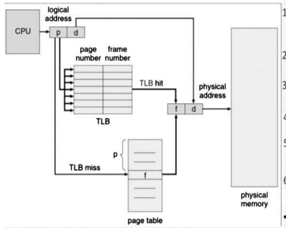
Module1(6)

TLB

- If the page size is high, we store the page table in the primary memory , therefore memory access time is increased.
- That is we need 2 memory access to fetch instruction from the memory. One is to access the page table and the other is to fetch actual instruction.



- The standard solution to this problem is to use a special, small, fast lookup hardware cache called a **translation look-aside buffer (TLB)**.
- The TLB is associative, high-speed memory.
- Each entry in the TLB consists of two parts : a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
- If the item is found, the corresponding value field is returned.
- The search is fast; the hardware, however , is expensive. Therefore the number of entries in a TLB is small.



- The TLB is used with page tables in the following way.
- The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB.
- If the page number is found, its frame number is immediately available and is used to access memory.
- If the page number is not in the TLB (known as a **TLB miss**), a memory reference to the page table must be made.
- When the frame number is obtained, corresponding changes are made in the TLB , so that they will be found next time very quickly.
- If the TLB is already full of entries, an existing entry must be selected for replacement. Replacement policies range from least recently used (LRU) through round-robin to random.

DEMAND PAGING

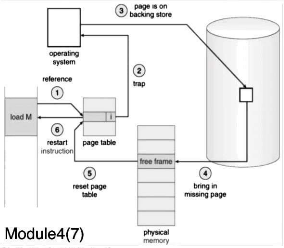
- A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk).
- When we want to execute a process, we swap it into memory.
- Any page execution is started on a page fault.
- In demand paging firstly no programs are in memory.
- When CPU generate an address, a page fault will occur. When a page fault occurs, we can load the entire program in to main memory or we can load only the needed program.

- Access to a page marked invalid causes a page fault.
- The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.
- The procedure for handling this page fault is straightforward

1. We check an internal table (usually kept with the process control block)for this process to determine whether the reference was a valid or aninvalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.
3. We find a free frame (by taking one from the free-frame list, for example).
4. We schedule a disk operation to read the desired page into the newlyallocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

- The efficiency of demand paging is increased by using locality of reference, because continuous hit is occurred. Hardware support for demand paging is

✓The page table must have valid/invalid bit
✓A swap area must need for performing swap out/swap in



Module4(7)

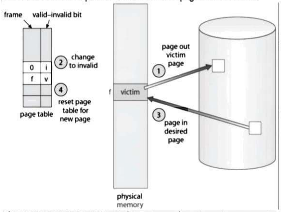
PAGING

- Paging is a memory management technique.
- In this approach, physical memory is divided in to fixed sized block called frames and logical memory is also divided in to the fixed sized blocks called pages.
- The size of the page is same as that of frame.
- The key idea of this method is to place the pages of a process in to the available frames of memory , whenever this process is to be executed.

PAGE REPLACEMENT

- Page replacement takes the following approach.

1. Find the location of the desired page on the **disk**.
2. Find a free frame:
   a. If there is a free frame, use it.
   b. If there is no free frame, use a page-replacement algorithm to select a **victim frame**.
   c . Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Continue the user process from where the page fault occurred.



❖PLACEMENT STRATEGY – It determines where in main memory to place the fetch program or job. Different placement strategies are

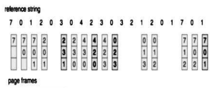
➢First fit :- The unused or free space in main memory is known as holes. A hole list is provided and it is in the form of linked list. The first fit places the program in the first storage hole which is large enough to hold it.

➢Best fit:- Best fit places the program in the tightest fitting hole. Here minimum waste of space is occurred.

➢Worst fit:- It places the program or data in the largest available hole that will hold it. Here more memory space wastage is occurred.

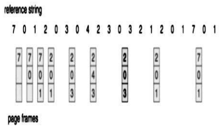
First-In-First-Out (FIFO) Algorithm

- ➢ A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- ➢ When a page must be replaced, the oldest page is chosen.
- ➢ Example:
  - o Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
  - o 3 frames (3 pages can be in memory at a time per process)



Optimal Page Replacement

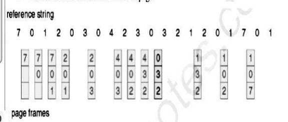
- ➢ Replace page that will not be used for longest period of time
- ➢ **It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly-hence known as OPT or MIN.**



- ➢ It is difficult to implement, because it requires future knowledge of the reference string

LRU Page Replacement

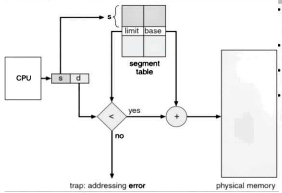
- ➢ Use past knowledge rather than future
- ➢ Replace page that has not been used in the most amount of time
- ➢ Associate time of last use with each page

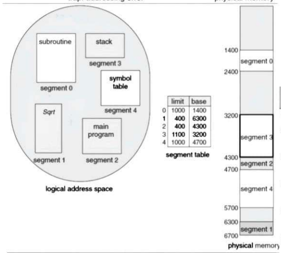


Module4(8)

SEGMENTATION

- Segmentation is a memory-management scheme that supports this programmer view of memory.
- Here each job is divided in to several segments of different sizes.
- Segments are variable size.
- So each segment has a base and limit.
- Limit is provided for avoiding segment overlapping.
- A program segment contains the program's main function, utility functions, data structures and so on.





FRAGMENTATION

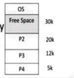
- As processes are loaded and removed from memory, the free memory space is broken in to little pieces.
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is called fragmentation.
- Two types of fragmentations are

❖EXTERNAL FRAGMENTATION

- External fragmentation is the unused area between two used areas. It is a serious problem. Here memory space to satisfy a request is available, but is not contiguous.

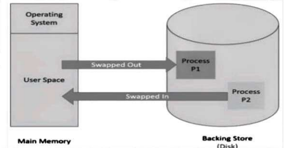
❖INTERNAL FRAGMENTATION

- Consider the following figure. Suppose a **25K** request is coming, then 30K is fully allocated because it is a fixed partition. Here 5k is wasted.
- That is the wasted space contained in a partition that is allocated for a request is called internal fragmentation.
- It is a wasted space with in a partition.
- Internal fragmentation can be reduced effectively assigning the smallest partition but large enough for the process.



SWAPPING

- Swapping is a mechanism in which a process can be swapped temporarily out of main memory or move to secondary storage(disk) and make that memory available to other processes.
- At some later time, the system swaps back the processes from the secondary storage to main memory.
- Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel
- Swapping may happen in the case of Round Robin scheduling. A process is swapped out when its time quantum finishes and later it is brought in to the memory for continued execution



LOGICAL ADDRESS & PHYSICAL ADDRESS

❖LOGICAL ADDRESS

- It is the virtual address generated by the CPU that can be viewed by the user
- Logical address is generated by the CPU during a program execution
- The logical address is virtual as it does not exist physically. Hence it is also called Virtual address
- This address is used as a reference to access the physical memory location(physical address)

➢Logical address space:- set of all logical addresses generated by the CPU in reference to a program is referred as logical address space.

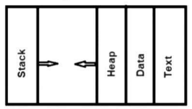
❖PHYSICAL ADDRESS

- Physical address is a location in a memory unit.
- The user can never view the physical address of program.
- The user cannot directly access the physical address. Instead, the physical address is accessed by its corresponding logical address by the user

➢Physical address space:- set of all physical addresses corresponding to the logical address is called physical address space.

Module4(9)

What does a process look like in memory?



Text Section: A Process, sometimes known as the Text Section, also includes the current activity represented by the value of the *Program Counter*.
Data Section: Contains the global variable.
Heap Section: Dynamically allocated memory to process during its run time.
Stack: The stack contains the temporary data, such as function parameters, returns addresses, and local variables.

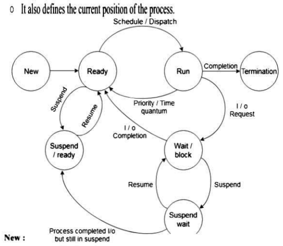
Process Control Block (PCB)

- o A Process Control Block is a data structure maintained by the Operating System for every process.
- o The PCB is identified by an integer process ID (PID).
- o There is a Process Control Block for each process, enclosing all the information about the process.
- o It is also known as the task control block. It is a data structure, which contains the following:

- o **Process ID:** Unique identification for each of the processes in the operating system.
- o **Process state:** A process can be new, ready, running, waiting, etc.
- o **Pointer:** A pointer to the parent process.
- o **Program counter:** The program counter lets you know the address of the next instruction, which should be executed for that process.
- o **CPU registers:** This component includes accumulators, index and general-purpose registers, and information of condition code.
- o **CPU scheduling information:** This component includes a process priority, pointers for scheduling queues, and various other scheduling parameters.
- o **Accounting and business information:** It includes the amount of CPU and time utilities like real time used, job or process numbers, etc.
- o **Memory-management information:** This information includes the value of the base and limit registers, the page, or segment tables. This depends on the memory system, which is used by the operating system.
- o **I/O status information:** This block includes a list of open files, the list of I/O devices that are allocated to the process, etc.

Process states

- o When a process executes, it passes through different states.
- o These stages may differ in different operating systems, and the names of these states are also not standardized.
- o A process state is a condition of the process at a specific instant of time.
- o It also defines the current position of the process.



New :
- ■ A program which is going to be picked up by the OS into the main memory is called a new process.

Ready :
- ■ Whenever a process is created, it directly enters in the ready state, in which it waits for the CPU to be assigned.
- ■ The OS picks the new processes from the secondary memory and puts all of them in the main memory.
- ■ The processes which are ready for the execution and reside in the main memory are called ready state processes.
- ■ There can be many processes present in the ready state.

Run :
- ■ One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm.
- ■ Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one.
- ■ If we have n processors in the system then we can have n processes running simultaneously.

Block or wait:
- ■ From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.
- ■ When a process waits for a certain resource to be assigned or for the input from the user then the OS moves this process to the block or wait state and assigns the CPU to the other processes.

Completion or termination:
- ■ When a process finishes its execution, it comes in the termination state.
- ■ All the context of the process (Process Control Block) will also be deleted and the process will be terminated by the Operating system.

Suspend ready:
- ■ A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspended ready state.
- ■ If the main memory is full and a higher priority process comes for the execution then the OS has to make room for the process in the main memory by throwing the lower priority process out into the secondary memory.

Module2(10)

What is a Thread?

- ■ A thread is a path of execution within a process.
- ■ A process can contain multiple threads.

Why Multithreading?

- ■ A thread is also known as a lightweight process.
- ■ The idea is to achieve parallelism by dividing a process into multiple threads.

Advantages of Thread over Process

- ■ *1. Responsiveness:* If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
- ■ *2. Faster context switch:* Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
- ■ *3. Effective utilization of a multiprocessor system:* If we have multiple threads in a single process, then we can schedule multiple threads on multiple processors. This will make process execution faster.
- ■ *4. Resource sharing:* Resources like code, data, and files can be shared among all threads within a process.(Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.)
- ■ *5. Communication:* Communication between multiple threads is easier, as the threads share common address space. While in process we have to follow some specific communication techniques for communication between two processes.
- ■ *6. Enhanced throughput of the system:* If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Bounded Buffer – Shared Memory Solution

* Shared data

```
#define BUFFER_SIZE 10
typedef struct
{
   ....
} item;
item buffer [BUFFER_SIZE];
int in = 0;
int out = 0;
```

The shared buffer is implemented as a circular array with two logical pointers : in & out.

The buffer is empty when in == out;
The buffer is full when ((in+1) % BUFFER_SIZE) == out)

Producer

```
item next_produced;
while (true)
{
   /* produce an item in next produced */
   while(((in+1) % BUFFER_SIZE) == out);
      /* do nothing */
   buffer [in] = next_produced;
   in = (in+1) % BUFFER_SIZE;
}
```

Consumer

```
item next_consumed;
while (true)
{
   while (in == out);
      /* do nothing */
   next_consumed = buffer [out];
   out = (out+1) % BUFFER_SIZE;
   /* consume item in next consumed */
}
```

INTER PROCESS COMMUNICATION

* Inter-process Communication (IPC) refers to the coordination of activities among cooperating processes.
* Two modes of IPC:
  - Shared Memory
  - Message Passing.

1. Shared Memory Systems.

* IPC communication using shared mly requires communicating processes to establish a region of shared memory.
* A shared mly region resides in the address space of the process creating the shared mly segment.
* Other processes that wish to communicate using this shared mly segment must attach it to their address space.
* They can then exchange Information by reading & writing data in the shared area.
* The communication is under the control of the user processes not the OS.

Module2(11)

2. Message Passing Systems

* Message Passing provides a mechanism to allow processes to communicate & to synchronize their actions without sharing the same address space.
* A message-passing facility provides 2 operations:
  - Send (message)
  - Receive (message).
* If processes P & Q want to communicate, they must send message to & receive message from each other.
  A communication link must exist b/w them.

Synchronization

Message passing may be either blocking or non-blocking also known as synchronous & asynchronous.

- Blocking Send – The sending process is blocked until the msg is received by the receiving process or by the mailbox.
- Non-Blocking Send – The sending process sends the msg & resumes operation.
- Blocking receive – The receiver blocks until a msg is available.
- Non-blocking Receive – The receiver retreives either a valid msg or a null.

5.2.1 File Attributes

- ➢ A file is referred to by its name.
- ➢ A file's attributes vary from one operating system to another but typically consist of these:

- o **Name:** The symbolic file name is the only information kept in human readable form.
- o **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- o **Type:** This information is needed for systems that support different types of files.
- o **Location:** This information is a pointer to a device and to the location of the file on that device.
- o **Size:** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- o **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- o **Time, date, and user identification:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

5.2.2 File Operations

- ➢ A file is an abstract datatype.
- ➢ To define a file properly, it needs to consider the operations that can be performed on files.

Creating a file: Two steps are necessary to create a file. First, space in the file system must be found for thefile.

Writing a file: To write a file, we make a system call specifying both the Name of the file and the information to be written to the file. The system must keep a write pointer to the location in the file where the next write is to take place.

- o **Reading a file:** To read from a file, we use a system call that specifies the name of

5.2.3 File Types

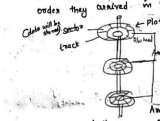
- ➢ The operating system should recognize and support file types. If an operating system recognizes the type of a file, it can then operate on the file in reasonable ways.
- ➢ The system uses the extension to indicate the type of the file and the type of operations that can be done on thatfile.

5.2.4 File Structure

- ➢ File types can be used to indicate the internal structure of the file.
- ➢ The operating requires that an executable file have a specific structure so that it can system termine where in memory to load the file and what the location of
- ➢ None - sequence of words, bytes
- ➢ Simple record structure
  - Lines
  - Fixed length
  - Variable length
- ➢ Complex Structures
  - Formatted document
  - Relocatable load file
- ➢ Can simulate last two with first method by inserting appropriate control characters

Module5(12)

1. FCFS Alg (First Come First Served)
   - Simplest of all disk scheduling alg.
   - In FCFS, ~~all~~ the requests are addressed in the order they arrived in the disk queue.



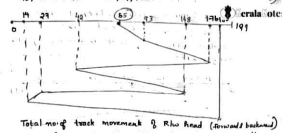
* Arm ~~is placed in~~ for read/write R/w head are placed in platters. From platter → surface → Track → Sector

Advantage (FCFS)
- Every request gets a fair chance.
- No indefinite postponment.

Disadvantage (FCFS)
- Does not try to optimize seek time.
- May not provide the best possible service.

Eg:- A disk contains 200 tracks (0-199)
- Request queue contains track no: 93, 176, 42, 148, 27, 14, 180
- Current position of R/w head = 55
- Calculate total no. of track movement of R/w head

93, 176, 42, 148, 27, 14, 180



Total no. of track movement of R/w head (forward / backward)
$= (176-55) + (176-42) + (148-42) + (148-14) + (180-14) = \underline{661}$

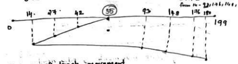
2. SSTF (Shortest Seek Time First)

* In SSTF, request having shortest seek time are executed first.
* So the seek time of every request is calculated in advance in queue & schedule according to their seek time.
* As a result, request near the disk arm will get executed first.

Advantages (SSTF)
* Avg response time will be decreased.
* Throughput increased.

Disadvantage (SSTF)
* Overhead to calculate the seek time in advance.
* Causes of starvation (waiting for long time) for a request if it has higher seek time.

Eg: Disk contains 200 tracks (0-199)
Request Queue – 93, 176, 42, 148, 27, 14, 180
Current position of R/w head – 55.
Calculate total no. of track movements by R/w head using SSTF.
[Shortest (nearest) seek time visited first]
[42, 93 - 42 is nearest]
from 14 - 93, 176, 148, 180



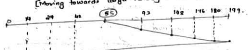
Total no. of track movement
= (55 – 14) + (180 – 14) = 207.

③ SCAN
* The arm moves in one direction only & satisfying all outstanding requests, until there is no more request in that direction.
* Service direction is then reversed.
* It works like an elevator.

Advantages
* High Throughput
* Low variance of response time
* Avg response time

Disadvantage.
* Long waiting time for request for locations just visited by disk arm.

Eg: Disk contains 200 tracks (0-199)
Request queue – 93, 176, 42, 148, 27, 14,
Current position of R/w head – 55
[Moving towards large value].

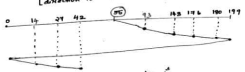


Total no. of track movement
= (199 – 55) + (199 – 14) = 329

④ CSCAN (Circular SCAN)
* It is restrict scanning to one direction only.
* When the last track has been visited in one direction, the arm is returned to opposite end of disk & scan begins again.

Advantages.
* Provides more uniform wait time compared to SCAN

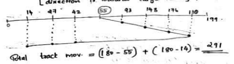
Eg: Disk contains 200 tracks (0-199)
Request queue – 93, 176, 42, 148, 27, 14, 180.
Current position of R/w head – 55.
[direction is towards large value].



Total no. of track movement
(199 – 55) + (199 – 0) + (42 – 0) = 385

⑤ LOOK
* Same as SCAN algorithm but arm goes only the final request.

Eg: Disk contains – 0 - 199 tracks.
Request queue – 93, 176, 42, 148, 27, 14, 180.
Current posn – 55.
[direction is towards large values].



Total track mov. = (180 – 55) + (180 – 14) = 291

⑥ CLOOK

Eg:

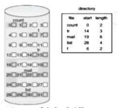
Total track mov = (180 – 55) + (180 – 14) + (42 – 14)
= 319

7.2 Linked Allocation

- With linked **allocation**, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
- The directory points to the first and last blocks of the file.
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks

5.7.1 Contiguous Allocation

- requires that each file occupy a set of contiguous blocks on disk.
- Disk addresses define a linear ordering on the disk.
- Best performance in most cases
- Simple – only starting location (block number) and length (number of blocks) are required
- Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line (downtime) or on-line**



Disk formatting

- A new magnetic disk is a blank slate: it is just a platter of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting,or physical formatting.
- Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer.
- The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC).
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area. When the sector is read, the ECC is recalculated and compared with the stored value. If the stored and calculated numbers are different, the data area of the sector has become corrupted and that the disk sector may be bad.
- The ECC is an error-correcting code. If only a few bits of data have been corrupted, ECC enable the controller to identify which bits have been changed and calculate what their correct values should be.

Module5(13)

The situation of the dining philosophers

- Philosophers spend their lives alternating thinking and eating
- Don't interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl
  - Need both to eat, then release both when done
- The shared data are
  semaphore chopstick[5];
- Where all the elements of chopstick are initialized to 1.
- The structure of Philosopher *i*:

```
do {
wait(chopstick[i]);
wait(chopstick[(i+1) % 5]);
...
/* eat for awhile */
...
signal(chopstick[i]);
signal(chopstick[(i+1) % 5]);
...
/* think for awhile */
... } while (true);
```

Semaphore Implementation

When a process executes the wait() operation and finds that the semaphore value is not positive, must wait. However, rather than engaging in busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which se another process to execute.

```
typedef struct{
int value;
struct process *list;
} semaphore;
```

- The wait() semaphore operation can be defined as

```
wait(semaphore *S) {
  S->value--;
  if (S->value < 0) {
    add this process to S->list;
    block();
  }
}
```

- the signal() semaphore operation can be defined as

```
signal(semaphore *S) {
  S->value++;
  if (S->value <= 0) {
    remove a process P from S->list;
    wakeup(P);
  }
```

SEMAPHORE

The hardware-based solutions to the critical-section problem presented are complicated for application programmers to use. To overcome this difficulty a synchronization tool called a semaphore is used. **A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait () and signal ()**. The wait () operation was originally termed P (from the Dutch *proberen*, "to test"); signal() was originally called V (from *verhogen*, "to increment"). The definition of wait () is as follows:

```
wait(S)
{
while S <= 0
// busy waiting
S--;
}
```

The definition of signal() is as follows:

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

CRITICAL SECTION

- 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.
- That is, no two processes are executing in their critical sections at the same time.

Critical Section Problem

- The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit section. The remaining code is the remainder section.
- The general structure of a typical process *Pi* is shown below

```
do
   {
        entry section
   critical section
        exit section
   remainder section
   } while (TRUE);
```

PETERSON'S SOLUTION

- A classic software-based solution to the critical-section problem known as Peterson's solution.
- Modern computer architectures perform basic machine-language instructions, such as load and store.
- There are no guarantees that Peterson's solution will work correctly on such architectures.
- We present the solution because it provides a good algorithmic description of solving the critical-section problem
- It illustrates some of the complexities involved in designing software that addresses the requirements of mutual exclusion, progress, and bounded waiting.
- Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections.
- The processes are numbered *Po* and P1. For convenience, when presenting *Pi*, we use *Pj* to denote the other process; that is, j equals 1 - i. Peterson's solution requires the two processes to share two data items:
  int turn;
  boolean flag[2];

The variable turn indicates whose turn it is to enter its critical section. That is, if turn == i, then process *Pi* is allowed to execute in its critical section. The flag array is used to indicate if a process *is ready* to enter its critical section. For example, if flag [i] is true, this value indicates that *Pi* is ready to enter its critical section. With an explanation of these data structures complete. The algorithm explains below.

Module3(14)