MODULE 1

FIRST 7 QUESTIONS ARE FROM MODEL PAPER

1) Define operating systems. Explain dual mode of os.

- An operating system is system software that acts as an intermediary between a user of a computer and the computer hardware.
- It is software that manages the computer hardware and allows the user to execute programs in a convenient and efficient manner.

The system can be assumed to work in two separate modes of operation:

1. User mode
2. Kernel mode (supervisor mode, system mode, or privileged mode).

- A hardware bit of the computer, called the mode bit, is used to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed by the operating system and one that is executed by the user.
- When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.

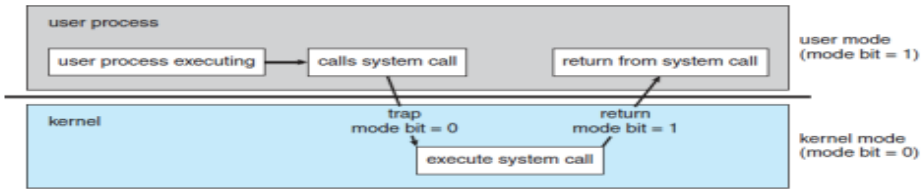


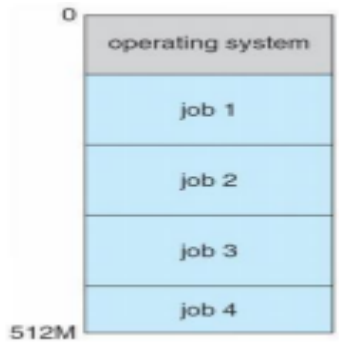
Figure Transition from user to kernel mode.

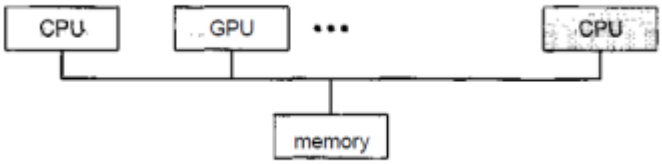
- At system boot time, the hardware starts in kernel mode.
- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the mode bit from 1 to 0).
- Thus, whenever the operating system gains control of the computer, it is in kernel mode.

---

2) Multiprogramming and multitasking (OPERATING SYSTEM STRUCTURE)

Feature	Multiprogramming	Multitasking
Definition	Increases CPU utilization by organizing jobs so the CPU always has one to execute.	Executes multiple jobs by switching among them so quickly that users can interact with each program.
User Interaction	Does not provide user interaction with the computer system.	Provides direct communication between the user and the system.
Switching Frequency	CPU switches to another job when the current job has to wait (e.g., for I/O).	CPU switches among jobs so frequently that users feel programs run simultaneously.
Response Time	No real-time interaction; response time is not a focus.	Requires short response time, typically less than one second.
System Perception	Jobs are executed in sequence based on resource availability.	Users feel like all programs are executed simultaneously.





---

3) Multiprocessor system and clustered system

Feature	Multiprocessor Systems	Clustered Systems
Definition	Systems with two or more processors in close communication, sharing bus, clock, memory, and peripherals.	Two or more individual systems connected via a network, sharing software resources.
Resource Sharing	Processors share memory and other hardware resources.	Systems share software resources and may store copies of files for high availability.
Communication	Communication between processors is through shared memory and hardware buses.	Communication between systems is through a network.
Reliability	Provides increased reliability using techniques like graceful degradation and fault tolerance.	Provides high availability by ensuring the service continues even if one or more systems fail.
Types	- Asymmetric Multiprocessing (Master/Slave) - Symmetric Multiprocessing (SMP, Peer processors).	- Asymmetric Clustering (Hot-standby mode) - Symmetric Clustering (All systems run applications and monitor each other).
Performance	Performance improvement depends on the coordination and overhead of shared resources.	Performance improvement depends on efficient network communication and clustering mechanisms.

---

4) OS services wrt users and programs.

OS provide services for the users of the system, including:

- User Interfaces - Means by which users can issue commands to the system. Depending on the operating system these may be a command-line interface , a Graphical User, or a batch command systems.
  In Command Line Interface (CLI)- commands are given to the system.
  In Batch interface – commands and directives to control these commands are put in a file and then the file is executed.
  In GUI systems- windows with pointing device to get inputs and keyboard to enter the text.

- Program Execution - The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.
- I/O Operations - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and files. For specific devices, special functions are provided (device drivers) by OS.
- File-System Manipulation – Programs need to read and write files or directories. The services required to create or delete files, search for a file, list the contents of a file and change the file permissions are provided by OS
- Communications - Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented by using the service of OS- like shared memory or message passing.
- Error Detection - Both hardware and software errors must be detected and handled appropriately by the OS. Errors may occur in the CPU and memory hardware, in I/O devices, and in the user program .

OS provide services for the efficient operation of the system, including:

- Resource Allocation – Resources like CPU cycles, main memory, storage space, and I/O devices must be allocated to multiple users and multiple jobs at the same time.
- Accounting – There are services in OS to keep track of system activity and resource usage, either for billing purposes or for statistical record keeping that can be used to optimize future performance.
- Protection and Security – The owners of information (file) in multiuser or networked computer system may want to control the use of that information. When several separate processes execute concurrently, one process should not interfere with other or with OS. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders must also be done, by means of a password.

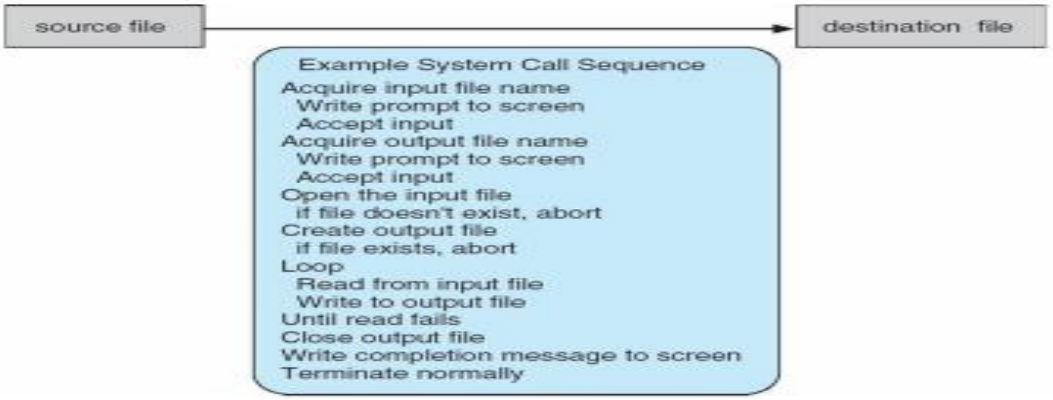
---

5) System calls and types

- System calls provides an interface to the services of the operating system. These are generally written in C or C++, although some are written in assembly for optimal performance.
- The below figure illustrates the sequence of system calls required to copy a file content from one file (input file) to another file (output file).

Example to Illustrate System Calls in File Copy Program

- **Write Message to Screen:** System call to display a prompt for the input filename.
- **Accept Input Filename:** System call to read the input filename.
- **Write Message to Screen:** System call to display a prompt for the output filename.
- **Accept Output Filename:** System call to read the output filename.
- **Open Input File:** System call to open the input file.
- **Open/Create Output File:** System call to open or create the output file.
- **Read and Write Data**
- **Close Files:** System calls to close both the input and output files.
- **Write Completion Message:** System call to display a success message.
- **Terminate Program:** System call to terminate the program normally.



Types of System calls:-

1. Process Control
    - Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
    - Processes must be created, launched, monitored, paused, resumed, and eventually stopped.
    - When one process pauses or stops, then another must be launched or resumed
    - Process attributes like process priority, max. allowable execution time etc. are set and retrieved by OS.
    - After creating the new process, the parent process may have to wait (wait time), or wait for an event to occur (wait event). The process sends back a signal when the event has occurred (signal event)
2. File Management The file management functions of OS are –
    - File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
    - After creating a file, the file is opened. Data is read or written to a file.
    - The file pointer may need to be repositioned to a point.
    - The file attributes like filename, file type, permissions, etc. are set and retrieved using system calls.
    - These operations may also be supported for directories as well as ordinary files.
3. Device Management
    - Device management system calls include request device, release device, read, write, reposition, get/set device attributes, and logically attach or detach devices.
    - When a process needs a resource, a request for resource is done. Then the control is granted to the process. If requested resource is already attached to some other process, the requesting process has to wait.
    - In multiprogramming systems, after a process uses the device, it has to be returned to OS, so that another process can use the device.
    - Devices may be physical , or virtual / abstract
4. Information Maintenance
    - Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
    - These system calls care used to transfer the information between user and the OS. Information like current time & date, no. of current users, version no. of OS, amount of free memory, disk space etc. are passed from OS to the user.
5. Communication

    • Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.

- The message passing model must support calls to:

1. Identify a remote process and/or host with which to communicate. o Establish a connection between the two processes.
2. Open and close the connection as needed.
3. Transmit messages along the connection.
4. Wait for incoming messages, in either a blocking or non-blocking state.
5. Delete the connection when no longer needed.

- The shared memory model must support calls to:

1. Create and access memory that is shared amongst processes (and threads.)
2. Free up shared memory and/or dynamically allocate it as needed.

- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data. It is easy to implement, but there are system calls for each read and write process.
- Shared memory is faster, and is generally the better approach where large amounts of data are to be shared. This model is difficult to implement, and it consists of only few system calls.

6. Protection
    - Protection provides mechanisms for controlling which users / processes have access to which system resources.
    - System calls allow the access mechanisms to be adjusted as needed, and for non-privileged users to be granted elevated access permissions under carefully controlled temporary circumstances.

---

6) Virtual machines

- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.
- Creates an illusion that a process has its own processor with its own memory.
- Host OS is the main OS installed in system and the other OS installed in the system are called guest OS.

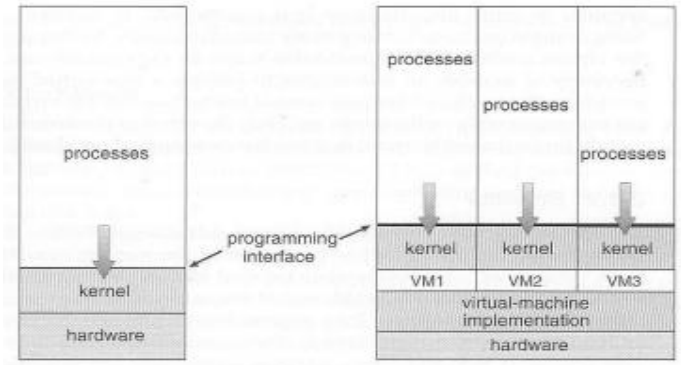
Implementation

- The virtual-machine concept is useful, it is difficult to implement.
- Work is required to provide an exact duplicate of the underlying machine. Remember that the underlying machine has two modes: user mode and kernel mode.
- The virtual-machine software can run in kernel mode, since it is the operating system. The virtual machine itself can execute in only user mode.

Benefits

- Virtual machines share the same hardware while running different execution environments (operating systems).
- The host system is protected from virtual machines, and virtual machines are isolated from one another. A virus in one guest OS does not affect others or the host.
- Resources can be shared using:
    - A shared file system volume.
    - A virtual communication network between virtual machines.
- Virtual machines allow separation of user program execution and system development, eliminating downtime during system changes and testing.
- Multiple operating systems can run simultaneously, enabling rapid code porting and testing in different environments.

- Two or more systems can run on a single hardware system.
  1. **Simulation:**
     Emulators translate instructions from guest system architecture to the host system's native instruction set, allowing cross-architecture program execution.
     Para-Virtualization:
  2. The guest OS runs on hardware that is similar but not identical to its preferred system, requiring modifications to the guest OS.

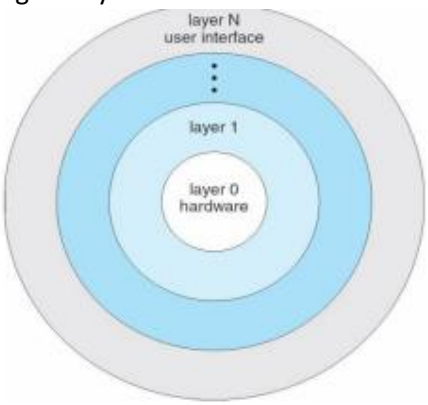


ıre: System modes. (A) Non-virtual machine (b) Virtual machine

---

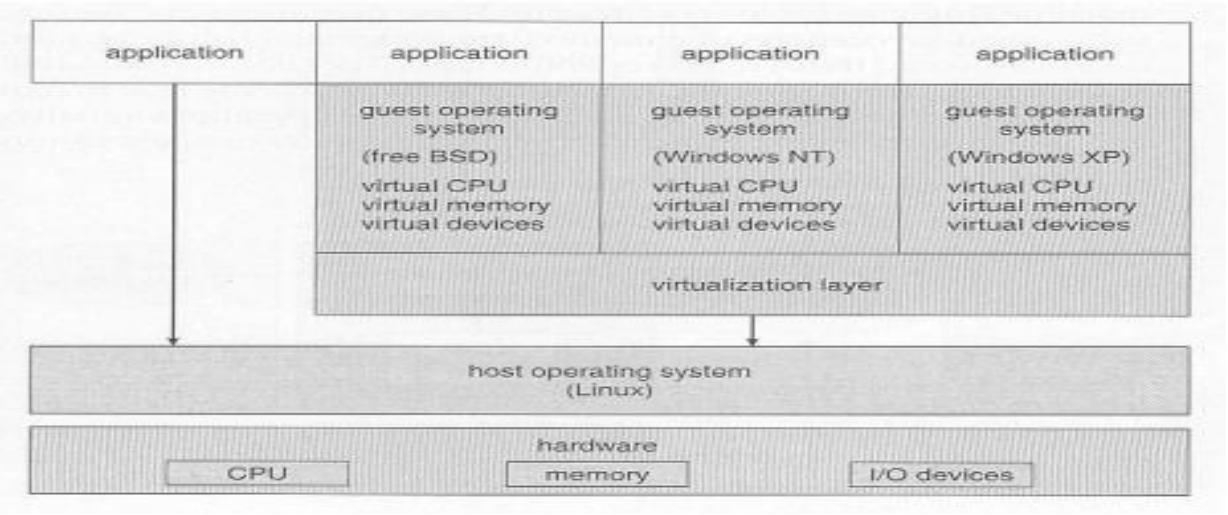
7) Layered approach for OS.

- The OS is broken into number of layers (levels). Each layer rests on the layer below it, and relies on the services provided by the next lower layer.
- Bottom layer (layer 0) is the hardware and the topmost layer is the user interface.
- A typical layer, consists of data structure and routines that can be invoked by higher-level layer.
- Advantage of layered approach is simplicity of construction and debugging.
- The layers are selected so that each uses functions and services of only lower-level layers. So simplifies debugging and system verification. The layers are debugged one by one from the lowest and if any layer doesn't work, then error is due to that layer only, as the lower layers are already debugged. Thus, the design and implementation are simplified.
- A layer need not know how its lower-level layers are implemented. Thus hides the operations from higher layers.



---

8) VM-Ware architechture

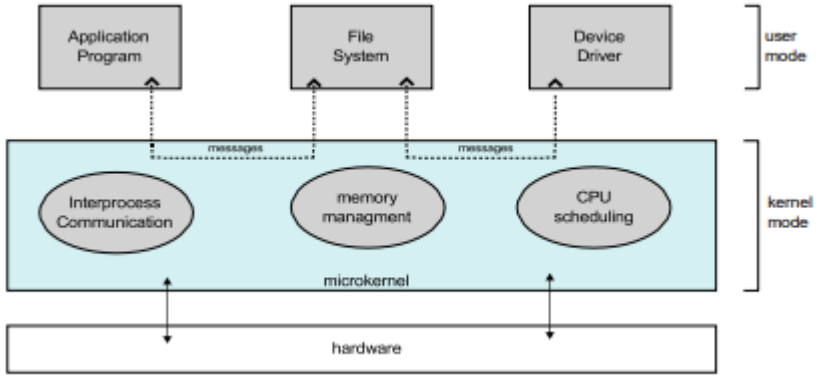


- VMware is a popular commercial application that abstracts Intel 80X86 hardware into isolated virtual machines. The virtualization tool runs in the user-layer on top of the host OS. The virtual machines running in this tool believe they are running on bare hardware, but the fact is that it is running inside a user-level application.
- VMware runs as an application on a host operating system such as Windows or Linux and allows this host system to concurrently run several different guest operating systems as independent virtual machines.
- In above scenario, Linux is running as the host operating system; FreeBSD, Windows NT, and Windows XP are running as guest operating systems. The virtualization layer is the heart of VMware, as it abstracts the physical hardware into isolated virtual machines running as guest operating systems. Each virtual machine has its own virtual CPU, memory, disk drives, network interfaces, and so forth.

---

9) Microkernels

- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs thus making the kernel as small and efficient as possible.
- The removed services are implemented as system applications.
- Most microkernels provide basic process and memory management, and message passing between other services.
- The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.



---

10) Process management and memory management

Memory Management

- Main memory is a large array of words or bytes. Each word or byte has its own address.
- Main memory is the storage device which can be easily and directly accessed by the CPU. As the program executes, the central processor reads instructions and also reads and writes data from main memory.
- To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.

The operating system is responsible for the following activities in connection with memory management:

- Keeping track of which parts of memory are currently being used by user.
- Deciding which processes and data to move into and out of memory.
- Allocating and deallocating memory space as needed

Process Management

- A program under execution is a process. A process needs resources like CPU time, memory, files, and I/O devices for its execution. These resources are given to the process when it is created or at run time. When the process terminates, the operating system reclaims the resources.
- The program stored on a disk is a passive entity and the program under execution is an active entity. A single-threaded process has one program counter specifying the next instruction to execute. The CPU executes one instruction of the process after another, until the process completes.

The operating system is responsible for the following activities in connection with process management:

- Scheduling process and threads on the CPU
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

MODULE 2

[FIRST 6 QUESTIONS ARE MODEL PAPER QUESTION]

1) Process and diff states of processes.

• A process is a program under execution. Its current activity is indicated by PC (Program Counter) and the contents of the processor's registers.

Process memory is divided into four sections as shown in the figure below:

Figure: Process in memory.

- The stack is used to store temporary data such as local variables, function parameters, function return values, return address etc.
- The heap which is memory that is dynamically allocated during process run time
- The data section stores global variables.
- The text section comprises the compiled program code.
- Note that, there is a free space between the stack and the heap. When the stack is full, it grows downwards and when the heap is full, it grows upwards.

Process State

A Process has 5 states. Each process may be in one of the following states –

1. New - The process is in the stage of being created.
2. Ready - The process has all the resources it needs to run. It is waiting to be assigned to the processor.
3. Running – Instructions are being executed.
4. Waiting - The process is waiting for some event to occur. For example, the process maybe waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
5. Terminated - The process has completed its execution

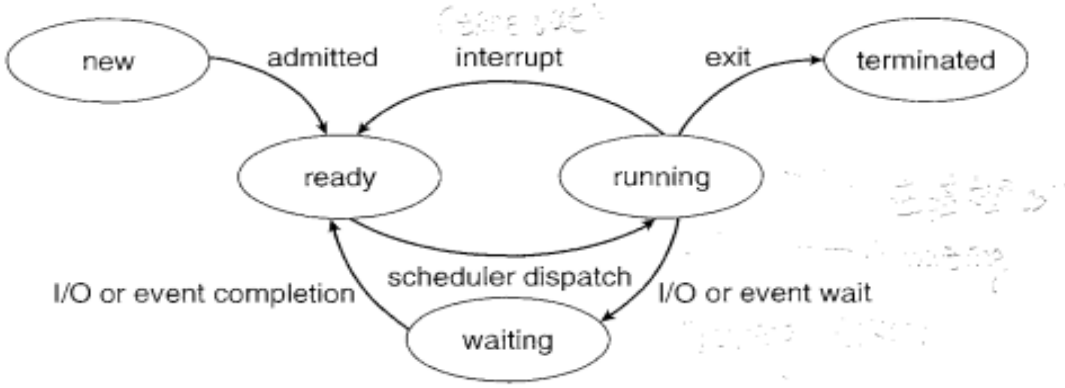


Figure 3.2 Diagram of process state.

Process Control Block

For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –

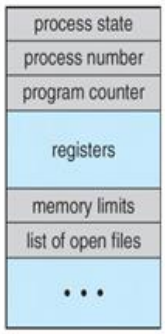


Figure: Process control block (PCB)

• Process State – The state of the process may be new, ready, running, waiting, 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. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

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

• Memory-management information – This includes information such as the value of the base and limit registers, the page tables, or the segment tables.

• 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.

2) Interprocess Communication. Direct and indirect communication wrt message passing

Interprocess Communication- Processes executing may be either co-operative or independent processes.

- Independent Processes – processes that cannot affect other processes or be affected by other processes executing in the system.
- Cooperating Processes – processes that can affect other processes or be affected by other processes executing in the system

a) Direct communication

- the sender and receiver must explicitly know each other's name.
- The syntax for send() and receive() functions are as follows-
  - send (P, message) – send a message to process P
  - receive(Q, message) – receive a message from process Q

Properties of communication link:

- A link is established automatically between every pair of processes that wants to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly one pair of communicating processes
- Between each pair, there exists exactly one link.

Types of addressing in direct communication –

- Symmetric addressing – the above-described communication is symmetric communication. Here both the sender and the receiver processes have to name each other to communicate.
- Asymmetric addressing – Here only the sender's name is mentioned, but the receiving data can be from any system.
  - send (P, message) --- Send a message to process P
  - receive (id, message). Receive a message from any process

b) Indirect communication uses shared mailboxes, or ports.

- A mailbox or port is used to send and receive messages. Mailbox is an object into which messages can be sent and received. It has a unique ID. Using this identifier messages are sent and received.
- Two processes can communicate only if they have a shared mailbox. The send and receive functions are –
  - send (A, message) – send a message to mailbox A
  - receive (A, message) – receive a message from mailbox A

Properties of communication link:

- A link is established between a pair of processes only if they have a shared mail
- A link may be associated with more than two processes
- Between each pair of communicating processes, there may be any number of links, each link is associated with one mailbox.
- A mail box can be owned by the operating system. It must take steps to –
  - create a new mailbox
  - send and receive messages from mailbox
  - delete mailboxes

---

3) Multithreaded programming.

A process having multiple threads of control, it can perform more than one task at a time. such a process is called multithreaded process/programming.

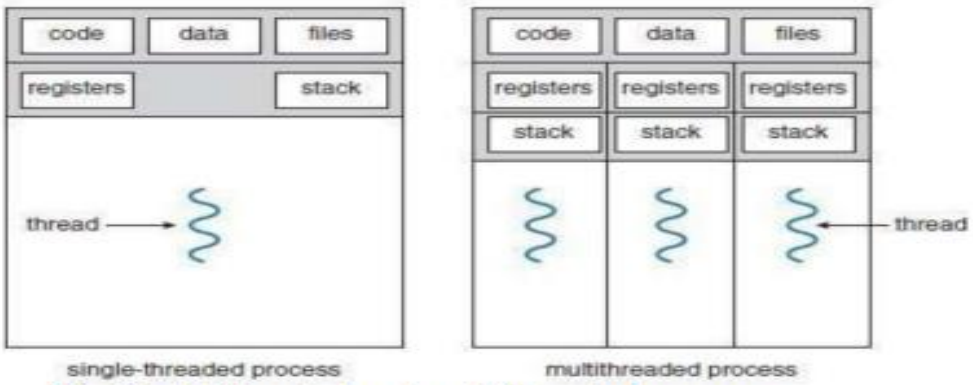


Fig: Single-threaded and multithreaded processes

Benefits of Multithreaded Programming

- Responsiveness A program may be allowed to continue running even if part of it is blocked. Thus, increasing responsiveness to the user.
- Resource Sharing By default, threads share the memory (and resources) of the process to which they belong. Thus, an application is allowed to have several different threads of activity within the same address-space.
- Economy Allocating memory and resources for process-creation is costly. Thus, it is more economical to create and context-switch threads.
- Utilization of Multiprocessor Architectures In a multiprocessor architecture, threads may be running in parallel on different processors. Thus, parallelism will be increased

---

4) Multithreading models

Support for threads may be provided at either

1. The user level, for user threads or
2. By the kernel, for kernel threads.

• User-threads are supported above the kernel and are managed without kernel support. Kernel-threads are supported and managed directly by the OS.

• Three ways of establishing relationship between user-threads &kernel-threads:

1. Many-to-one model
2. One-to-one model and
3. Many-to-many model.

Many-to-One Model

- Many user-level threads are mapped to one kernel thread.

  Advantages:
- Thread management is done by the thread library in user space, so it is efficient.

  Disadvantages:
- The entire process will block if a thread makes a blocking system-call.
- Multiple threads are unable to run in parallel on multiprocessors.

For example: Solaris green threads ,GNU portable threads.

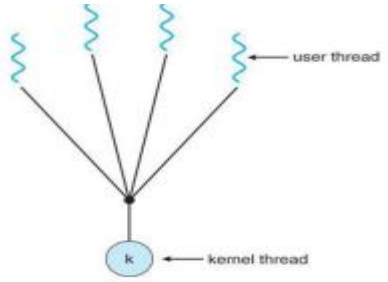


Fig: Many-to-one model

One-to-One Model

• Each user thread is mapped to a kernel thread.

Advantages:

- It provides more concurrency by allowing another thread to run when a thread makes a blocking system-call.
- Multiple threads can run in parallel on multiprocessors.

Disadvantage:

- Creating a user thread requires creating the corresponding kernel thread.

• For example: Windows NT/XP/2000, Linux

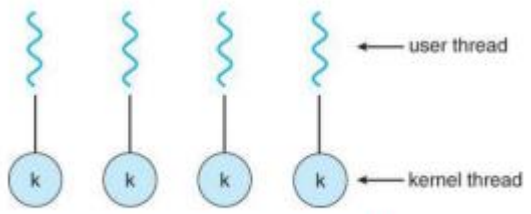


Fig: one-to-one model

Many-to-Many Model

• Many user-level threads are multiplexed to a smaller number of kernel threads.

Advantages:

- Developers can create as many user threads as necessary
- The kernel threads can run in parallel on multiprocessor.
- When a thread performs a blocking system-call, kernel can schedule another thread for execution.

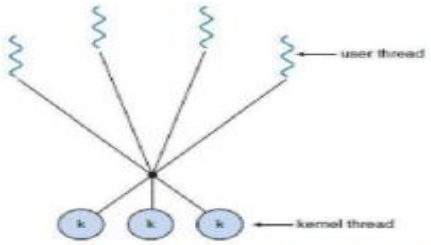


Fig: Many-to-many model

---

5) Difference between Shared memory and message passing.

Sl No	**Shared Memory**	**Message passing**
1.	A region of memory is shared by communicating processes, into which the information is written and read	Message exchange is done among the processes by using objects.
2.	Useful for sending large block of data	Useful for sending small data.
3.	System call is used only to create shared memory	System call is used during every read and write operation.
4.	Message is sent faster, as there are no system calls	Message is communicated slowly.

• Shared Memory is faster once it is set up, because no system calls are required and access occurs at normal memory speeds. Shared memory is generally preferable when large amounts of information must be shared quickly on the same computer.

• Message Passing requires system calls for every message transfer, and is therefore slower, but it is simpler to set up and works well across multiple computers. Message passing is generally preferable when the amount and/or frequency of data transfers is small.

---

6) Scheduling Criteria.

1. **CPU utilization**: We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from O to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

2. **Throughput**: If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.

3. **Turnaround time**: This is the important criterion which tells how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing 1/0.

4. **Waiting time**: The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does 1/0, it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

5. **Response time**: In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of

---

7) Threads. Threads vs processes. Threading issues.

Threads

- A thread is a basic unit of CPU utilization.
- It consists of
  - thread ID
  - PC
  - register-set
  - stack.
- It shares with other threads belonging to the same process its code-section &data-section.
- A traditional (or heavy weight) process has a single thread of control.

Threads vs Processes

Feature	Process	Thread
Definition	Independent program execution	Lightweight unit in a process
Memory	Separate memory space	Shared memory within a process
Overhead	High	Low
Communication	Complex (IPC needed)	Easy (shared memory)
Isolation	Strong	Weak
Use Case	Independent apps	Tasks within a single app

Threading issues

1) fork() and exec() System-calls

- fork() is used to create a separate, duplicate process.
- If one thread in a program calls fork (),then

1. Some systems duplicates all threads and
2. Other systems duplicate only the thread that invoked the fork O.

- If a thread invokes the exec(), the program specified in the parameter to exec() will
- replace the entire process including all threads.

2) Thread Cancellation

- This is the task of terminating a thread before it hascompleted.
- Target thread is the thread that is to be cancelled
- Thread cancellation occurs in two different cases:
    1. Asynchronous cancellation: One thread immediately terminates the target thread.
    2. Deferred cancellation: The target thread periodically checks whether it should be
    terminated.

3) Signal Handling

- In UNIX, a signal is used to notify a process that a particular event has occurred.
- All signals follow this pattern:

1. A signal is generated by the occurrence of a certain event.
2. A generated signal is delivered to a process.
3. Once delivered, the signal must be handled.

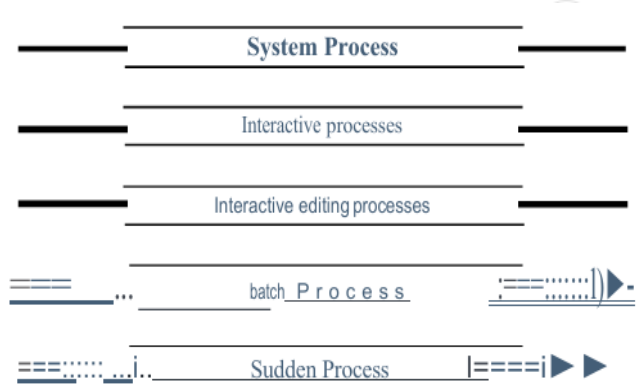
- A signal handler is used to process signals.
- A signal may be received either synchronously or asynchronously, depending on the source.

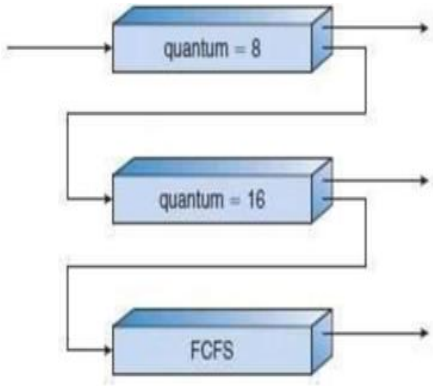
1. Synchronous signals
    - Delivered to the same process that performed the operation causing the signal.
    - E.g. illegal memory access and division by 0.
2. Asynchronous signals
    - Generated by an event external to a running process.
    - E.g. user terminating a process with specific keystrokes <ctrl> <c>.

- Every signal can be handled by one of two possible handlers:
    1. A Default Signal Handler - Run by the kernel when handling the signal.
    2. A User-defined Signal Handler -Overrides the default signal handler.

8) Multilevel queue Scheduling vs Multilevel feedback queue scheduling.

Aspect	Multilevel Queue Scheduling	Multilevel Feedback Queue Scheduling
Process Movement	Processes are permanently assigned to one queue based on specific properties.	Processes can move between queues based on their behavior or performance.
Basis of Classification	Processes are classified by properties such as memory size, priority, or type.	Processes are dynamically reclassified based on their CPU burst characteristics.
Scheduling Between Queues	Commonly fixed-priority preemptive scheduling.	Scheduling depends on dynamic adjustments to process priority (e.g., aging).
Aging Prevention	Starvation may occur if lower-priority queues are indefinitely blocked.	Aging is implemented to prevent starvation by promoting processes in lower-priority queues.
Queue Scheduling Algorithm	Each queue has its own fixed scheduling algorithm (e.g., FCFS, RR).	Queues have flexible scheduling rules that adapt based on CPU usage and waiting time.
Example of Use	Suitable for systems with clearly separated process types (e.g., interactive vs. batch).	Suitable for systems where process behavior varies and needs dynamic prioritization.
Complexity	Simpler to implement due to fixed queue assignments.	More complex due to dynamic process movement and aging mechanisms.
CPU Time Allocation	Each queue may get a fixed time slice (e.g., 80% for foreground, 20% for background).	CPU allocation adapts as processes shift between queues based on their performance.





---

9) Explain context switching.

• The task of switching a CPU from one process to another process is called context switching. Context-switch times are highly dependent on hardware support (Number of CPU registers).
• Whenever an interrupt occurs (hardware or software interrupt), the state of the currently running process is saved into the PCB and the state of another process is restored from the PCB to the CPU.
• Context switch time is an overhead, as the system does not do useful work while switching.

MODULE 3

[FIRST 5 ARE MODEL PAPER QUESTIONS]

1) Deadlock.

A process requests resources, if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a Deadlock.

Conditions for Deadlock:

1. Mutual exclusion: At least one resource must be held in a non-sharable mode, that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3. No preemption: Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait: A set {Po, P1, ••• , Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ••. , Pn-I is waiting for a resource held by Pn and Pn is waiting for a resource held by P0

Recovery from a Deadlock:

The system recovers from the deadlock automatically. There are two options for breaking a deadlock one is simply to abort one or more processes to break the circular wait. The other is to preempt some resources from one or more of the deadlocked processes.

1) Process Termination

To eliminate deadlocks by aborting a process, use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

1. **Abort all deadlocked processes:** This method clearly will break the deadlock cycle, but at great expense; the deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.
2. **Abort one process at a time until the deadlock cycle is eliminated**: This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

2) Resource Preemption

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. **Selecting a victim.**
   we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed during its execution.
2. **Rollback**
   We must roll back the process to some safe state and restart it from that state. Since it is difficult to determine what a safe state is, the simplest solution is a total rollback: abort the process and then restart it.
3. **Starvation**

2) Peterson’s solution for critical section problem.

Peterson's solution is a classic software-based approach to solving the critical-section problem for two processes. While it provides a theoretical framework for ensuring mutual exclusion, progress, and bounded waiting, it may not function correctly on modern computer architectures due to reordering and caching optimizations.

Key Elements:

- **Processes:** Two processes P0 and P1 (or Pi and Pj, where j=1−i).
- **Shared Variables:**
    1. int turn: Indicates whose turn it is to enter the critical section. If turn == i, Pi is allowed.
    2. boolean flag[2]: Indicates whether a process is ready to enter its critical section. If flag[i] == true, Pi is ready.

Structure of Process PiP_iPi:

```
do {
    flag[i] = TRUE;      // Indicate readiness
    turn = j;            // Allow the other process to proceed if
  while (flag[j] && turn == j)
        ; // Busy wait

    // Critical Section
    flag[i] = FALSE;     // Exit protocol
    // Remainder Section
} while (TRUE);
```

1. **Entry Protocol:**
    - Process Pi sets flag[i] = TRUE to signal readiness.
    - It sets turn = j, indicating it yields priority to Pj.
    - Pi waits in a loop if Pj is also ready (flag[j] == TRUE) and has priority (turn == j).
2. **Critical Section Access:**
    - Only one process enters the critical section at a time based on the above checks.
3. **Exit Protocol:**
    - After exiting the critical section, Pi sets flag[i] = FALSE to signal it is no longer ready.

Correctness of Peterson's Solution

1. **Mutual Exclusion:**
    - Mutual exclusion is guaranteed as PiP_iPi can enter the critical section only if:
        - flag[j]==FALSE flag[j] == FALSE flag[j]==FALSE (i.e., Pj is not ready), or
        - turn == i (i.e., Pi has priority).
    - If both processes attempt to enter simultaneously, only one assignment to turn (either i or j) will persist. This ensures only one process enters the critical section at a time.
2. **Progress:**
    - If Pj is not ready (flag[j] == FALSE), Pi can proceed without delay.
    - If both processes are ready, the process with the priority (turn) will proceed. The other will wait until the first process exits the critical section and resets its flag.
3. **Bounded Waiting:**
    - Pi can be delayed only if Pj is in the critical section or ready to enter (flag[j] == TRUE and turn == j).
    - Once Pj exits its critical section, it resets flag[j] = FALSE, allowing Pi to proceed.
    - If Pj becomes ready again, it must set turn = i, ensuring Pi is served before Pj re-enters.

3) Semaphore. Dining Philosopher problem.

A semaphore is a synchronization tool is used solve various synchronization problem and can be implemented efficiently.

- Semaphore do not require busy waiting.
- A semaphore S is an integer variable that is accessed only through two standard atomic operations: wait () and signal (). The wait () operation was originally termed P and signal() was called V.

Definition of wait():

```
wait (S) {
while S <= 0
; // no-op
S--
```

Definition of signal():

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

Dining philosopher's problem

Five philosophers sit at a circular table with a bowl of rice in the center. They alternate between thinking and eating. Each philosopher must pick up two chopsticks to eat, but only one chopstick can be picked up at a time. The challenge is to manage resource allocation (chopsticks) to prevent deadlock and starvation.

Solution Using Semaphores:

- **Chopsticks Representation:**
  - Each chopstick is represented by a semaphore.
  - semaphore chopstick[5]; (initialized to 1 for all elements).
- **Philosopher's Actions:**
  - To pick up a chopstick: execute wait() on the corresponding semaphore.
  - To release a chopstick: execute signal() on the corresponding semaphore.

Deadlock Prevention Strategies:

1. **Limit the Number of Philosophers:**
   - Allow at most four philosophers to sit simultaneously at the table.
   - This ensures at least one philosopher can eat at any time, avoiding circular wait.
2. **Check Availability Before Picking Up:**
   - A philosopher picks up chopsticks only if both are available.
   - This prevents a philosopher from holding one chopstick indefinitely while waiting for the other.
3. **Asymmetric Approach:**
   - Odd-numbered philosophers pick up their left chopstick first and then their right.
   - Even-numbered philosophers pick up their right chopstick first and then their left.
   - This breaks the symmetry and prevents circular wait.

```
while (true)
{
        wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );
        // eat
        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );
        // think
}
```

4) Resource allocation graph.

Deadlocks can be described in terms of a directed graph called System Resource-Allocation Graph
A **resource-allocation graph** is used to represent processes and resource types in a system. It helps in analyzing resource allocation and detecting potential deadlocks.

Components of a Resource-Allocation Graph:

1. **Vertices:**
   - **Processes (P):** Represented as circles. $P = \{P_1, P_2, ..., P_n\}$.
   - **Resources (R):** Represented as rectangles. $R = \{R_1, R_2, ..., R_m\}$.
     - Each instance of a resource is shown as a dot inside the rectangle.
2. **Edges:**
   - **Request Edge ($P_i \rightarrow R_j$):** Directed edge from a process P_i to a resource R_j, indicating that P_i has requested an instance of R_j.
   - **Assignment Edge ($R_j \rightarrow P_i$):** Directed edge from a resource R_j to a process P_i, indicating that an instance of R_j is allocated to P_i.

Graph Dynamics:

1. **Request:**
   - When P_i requests R_j, a request edge $P_i \rightarrow R_j$ is added.
2. **Allocation:**
   - If the request is fulfilled, the request edge is transformed into an assignment edge $R_j \rightarrow P_i$.
3. **Release:**
   - When P_i releases R_j, the assignment edge is removed.

Example Graph Analysis:

Graph Details:

- **Processes:** $P = \{P_1, P_2, P_3\}$.
- **Resources:** $R = \{R_1, R_2, R_3, R_4\}$.
- **Edges:** $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$.
- **Resource Instances:**
  $R_1 = 1, R_2 = 2, R_3 = 1, R_4 = 3$.

Process States:

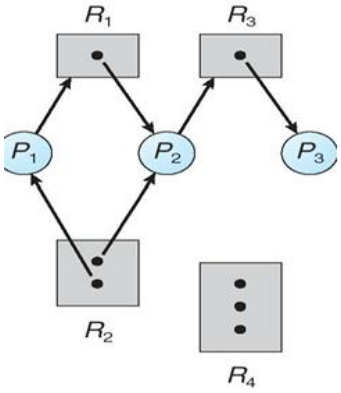
1. P_1: Holding R_2, waiting for R_1.
2. P_2: Holding R_1 and R_2, waiting for R_3.
3. P_3: Holding R_3.

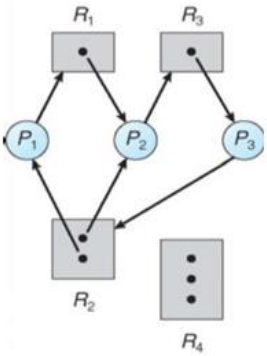
Cycle and Deadlock Analysis:

1. **Cycle Detection:**
   - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$.
   - $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$.
2. **Deadlock Detection:**
   - **Deadlock Exists:** If each resource in the cycle has only one instance, processes P_1, P_2, and P_3 are deadlocked.
     - P_1 waits for R_1 held by P_2.
     - P_2 waits for R_3 held by P_3.
     - P_3 waits for R_2, which P_1 or P_2 holds.

Cycle Without Deadlock:

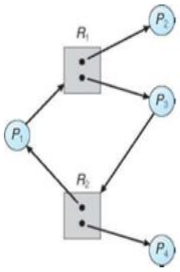
- If a resource in the cycle has multiple instances, a cycle may not imply deadlock.
- Example: If P_4 releases R_2, P_3 can acquire R_2, breaking the cycle.





urce-allocation graph with a deadlock.

llocation graph with a cycle but no deadlock



Key Observations:

1. **Cycle as a Necessary Condition:**
   - A cycle is a necessary condition for deadlock.
   - **Single-instance resources:** A cycle implies deadlock.
   - **Multiple-instance resources:** A cycle does not necessarily imply deadlock.
2. **Deadlock Resolution:**
   - Releasing resources (e.g., P4P_4P4 releasing R2R_2R2) can resolve the deadlock by breaking the cycle.

---

5) Critical Section problem and their requirements.

• Consider a system consisting of n processes {Po, Pl, ... ,Pn-1}.
• Each process has a segment of code, called a critical section in which the process may be changing common variables, updating a table, writing a file, and soon
• 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 sometime.
• The critical-section problem is to design a protocol that the processes can use to cooperate.

A solution to the critical-section problem must satisfy the following three requirements:

1. **Mutual exclusion**: If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress**: If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. **Bounded waiting**: There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

---

6) Readers writers problem.

Readers-Writers Problem

• A data set is shared among a number of concurrent processes
• Readers - only read the data set; they do not perform any updates
• Writers - can both read and write.
• Problem - allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time.

- Shared Data
- Dataset
- Semaphore mute initialized to 1.
- Semaphore reinitialized to1.
- Integer read count initialized to 0.

```
while (true)
{
        wait (wrt) ;
        // writing is performed
        signal (wrt) ;
}
```

The structure of a writer process

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

The structure of a reader process

---

7) Bounded buffer problem.

• N buffers, each can hold one item
• Semaphore mutex initialized to the value 1
• Semaphore full initialized to the value 0
• Semaphore empty initialized to the value N.

```
while (true)
{
        // produce an item
        wait (empty);
        wait (mutex);
        // add the item to the buffer
        signal (mutex);
        signal (full);
}
```

The structure of the producer process:

```
while (true)
{
        wait (full);
        wait (mutex);
        // remove an item from buffer
        signal (mutex);
        signal (empty);
        // consume the removed item
}
```

The structure of the consumer process:

---

8) Deadlock prevention and Deadlock avoidance (bankers Algorithm).

The Banker's Algorithm is a deadlock avoidance algorithm used in resource allocation systems where multiple instances of each resource type exist. It ensures the system remains in a **safe state** by checking resource availability before allocating resources to processes.

Key Concepts

1. **Safe State:**
   - A system is in a safe state if there exists a sequence of processes (P1,P2,...,Pn)(P_1, P_2, ..., P_n)(P1,P2,...,Pn) such that each process can complete its execution with the currently available resources and the resources held by other processes after they complete.
2. **Unsafe State:**
   - A system is in an unsafe state if it is possible for a deadlock to occur but has not necessarily occurred yet.

Required Data Structures

1. **Available Vector (size m):**
   - Tracks the number of available instances of each resource type.
   - Example: If $\text{Available}[j] = k$, then k instances of resource type R_j are available.
2. **Max Matrix ($n \times m$):**
   - Specifies the maximum demand of each process.
   - Example: $\text{Max}[i][j] = k$ means process P_i may request at most k instances of resource type R_j.
3. **Allocation Matrix ($n \times m$):**
   - Indicates the number of instances of each resource type currently allocated to each process.
   - Example: $\text{Allocation}[i][j] = k$ means k instances of R_j are allocated to P_i.
4. **Need Matrix ($n \times m$):**
   - Indicates the additional resources each process requires to complete its task.
   - Calculated as:

$$\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$$

Steps in Banker's Algorithm

1. **Initialization:**
   - All data structures (Available, Max, Allocation, and Need) are initialized.
2. **Resource Request:**
   - When a process PiP_iPi requests resources, the algorithm checks:
     - If the request does not exceed the process's **maximum need**.
     - If the requested resources are available.
   - If the above conditions are satisfied, the system proceeds to a **safety check**.
3. **Safety Check:**
   - Temporarily allocate the resources to PiP_iPi and simulate the execution of all processes.
   - If a safe sequence of processes exists, the resources are officially allocated.
   - Otherwise, the allocation is rolled back, and PiP_iPi must wait.
4. **Completion:**
   - Once a process completes, it releases its resources, updating the **Available** vector.

Notation Simplification

1. **Vectors Comparison:**
   - $X \leq Y$: Each element of $X[i] \leq Y[i]$.
   - Example: $(1, 7, 3, 2) \leq (2, 8, 4, 3)$.
2. **Allocation and Need Rows:**
   - Allocation_i: Resources allocated to P_i.
   - Need_i: Remaining resources required by P_i.

Deadlock prevention ensures that at least one of the four necessary conditions for deadlock does not hold, thereby avoiding deadlock entirely. Below is a breakdown of the conditions and strategies to prevent them:

---

1. Mutual Exclusion

- **Definition:** Some resources can only be used by one process at a time (non-sharable resources).
- **Prevention:**
  - Use sharable resources wherever possible (e.g., read-only files).
  - **Limitations:**
    - Certain resources, like printers or hardware devices, are inherently non-sharable, so mutual exclusion cannot always be avoided.

---

2. Hold and Wait

- **Definition:** A process holds at least one resource and is waiting to acquire additional resources.
- **Prevention Strategies:**
  - **Protocol 1:** Require a process to request all required resources at the start of execution.
    - **Disadvantage:** Low resource utilization since resources may be held for longer than necessary.
  - **Protocol 2:** Allow a process to hold some resources but release them before requesting new ones.
    - **Disadvantage:** Starvation may occur if a process frequently needs to release and reacquire resources.

Example:
A process copying data from a DVD to disk and then printing results:

- Protocol 1: Requests DVD drive, disk file, and printer at the start. Resources remain idle when not in use.
- Protocol 2: Requests DVD drive and disk file initially, then releases them before requesting the printer.

---

3. No Preemption

- **Definition:** Allocated resources cannot be forcibly taken away from a process.
- **Prevention Strategies:**
  - If a process requests a resource that is unavailable, preempt all its currently held resources.
    - Add the preempted resources to the list of resources it is waiting for.
    - Restart the process only when all requested and preempted resources are available.
  - If resources are unavailable but held by another waiting process, preempt those resources and allocate them to the requesting process.

Example Workflow:

1. Process P1P_1P1 requests a resource.
2. If unavailable:
    - Check if it is allocated to another process P2P_2P2 waiting for additional resources.
    - Preempt P2P_2P2's resources, allocate them to P1P_1P1, and add P2P_2P2 to the waitlist.
3. Process P1P_1P1 can only proceed when all its requested and preempted resources are available.

---

4. Circular Wait

- **Definition:** A circular chain of processes exists, where each process holds a resource and waits for another resource held by the next process in the chain.
- **Prevention Strategy:** Impose a total order on resource types and require processes to request resources in a strictly increasing order.
    - **Implementation:**
        - Assign a unique integer to each resource type.
        - Processes can request resources only in the order of these integers.

Example:

If resource types include tape drives, disk drives, and printers:

- Assign F(tape drive)=1F(\text{tape drive}) = 1F(tape drive)=1, F(disk drive)=5F(\text{disk drive}) = 5F(disk drive)=5, F(printer)=12F(\text{printer}) = 12F(printer)=12.
- A process may request a tape drive, then a disk drive, and finally a printer, but not in reverse order.

MODULE 4

[FIRST 5 QUESTION ARE FROM MODEL PAPER]

1) What is TLB? Its working.

Translation Look aside Buffer

• A special, small, fast lookup hardware cache, called a translation look-aside buffer (TLB).
• 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. Typically, the number of entries in a TLB is small, often numbering between 64 and 1,024.
• The TLB contains only a few of the page-table entries.

Working:

• When a logical-address is generated by the CPU, its page-number is presented to the TLB.
• If the page-number is found (TLB hit), its frame-number is immediately available and used to access memory
• If page-number is not in TLB (TLB miss), a memory-reference to page table must be made. The obtained frame-number can be used to memory
• In addition, we add the page-number and frame-number to the TLB, so that they will be found quickly on the next reference.
• If the TLB is already full of entries, the OS must select one for replacement.
• Percentage of times that a particular page-number is found in the TLB is called hit ratio.

Advantage: Search operation is fast. Disadvantage: Hardware is expensive.

• Some TLBs have wired down entries that can't be removed.
• Some TLBs store ASID (address-space identifier) in each entry of the TLB that uniquely identify each process and provide address space protection for that process.

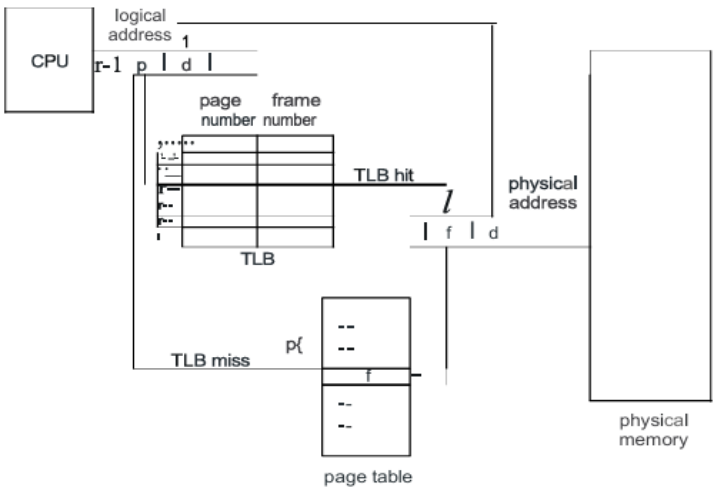


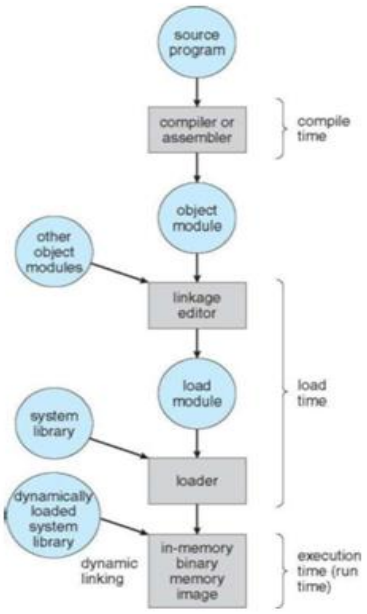
Figure 1: Paging hardware with TLB

---

2) Address binding.

Address binding of instructions to memory-addresses can happen at 3 different stages.

1. **Compile Time** - If it is known at compile time where a program will reside in physical memory, then absolute code can be generated by the compiler, containing actual physical addresses. However, if the load address changes at some later time, then the program will have to be recompiled.
2. **Load Time** - If the location at which a program will be loaded is not known at compile time, then the compiler must generate relocatable code, which references addresses relative to the start of the program. If that starting address changes, then the program must be reloaded but not recompiled.
3. **Execution Time** - If a program can be moved around in memory during the course of its execution, then binding must be delayed until execution time.



---

3) Demand paging.
Demand paging is a technique used in **virtual memory systems** to load only the pages of a program into physical memory when they are actually needed, instead of loading the entire program at once. This technique is efficient because it avoids loading unnecessary portions of a program into memory, thus conserving memory resources and improving overall system performance.
Key Concepts

1. **Lazy Swapper/Pager**: In demand paging, a **pager** is responsible for loading individual pages when needed. Unlike a traditional swapper, which loads entire processes, a pager focuses on individual pages.
2. **Page Table**: The **page table** is a critical component in demand paging, maintaining a record of pages that are loaded into memory. Each page table entry has a **valid-invalid bit**

indicating whether the page is in memory (valid) or not (invalid). If the page is invalid, it could either not be in the address space or be currently on disk.

3. **Page Faults**: When a process tries to access a page that is not currently in memory, a **page fault** occurs. The operating system then handles the fault by loading the required page into memory. This involves several steps:
    - Trap to the OS.
    - Check the page reference (valid or invalid).
    - If the page is invalid, terminate the process. If valid but not in memory, load the page.
    - Once the page is loaded, the process continues execution as if the page was always in memory.
4. **Handling of Page Faults**: The system may initiate multiple steps to bring the page into memory, such as:
    - Saving the process state.
    - Locating the page on the disk.
    - Loading the page into an available frame in memory.
    - Updating the page table.
    - Restarting the interrupted instruction.
5. **Performance Considerations**: Demand paging can significantly impact system performance. The **effective access time** is affected by the probability of page faults (denoted by **p**). A higher page fault rate leads to higher effective access time and slower process execution. The formula for effective access time is:

Effective Access Time=(1−p)×ma+p×page fault time\text{Effective Access Time} = (1 - p) \times ma + p \times \text{page fault time}Effective Access Time=(1−p)×ma+p×page fault time
Where:

- **ma** is the memory access time.
- **p** is the probability of a page fault.
- **page fault time** is the time it takes to service a page fault (usually much higher than a normal memory access).

6. **Swap Space**: **Swap space** is used to store pages that are not in memory. The swap space is typically located on a disk, which is slower than main memory. Efficient management of swap space can improve demand paging performance by minimizing the time spent on disk I/O operations.
7. **Optimization**: Demand paging works best when the page fault rate is kept low. If a program accesses several pages per instruction or has a high page fault rate, it can lead to significant performance degradation.

Key Benefits of Demand Paging

- **Efficient memory usage**: Only the pages that are required by the process are loaded, saving memory.
- **Faster program execution**: By loading pages only when needed, the system avoids wasting time and memory on unused code.
- **Better multiprogramming**: Since not all pages need to be in memory at once, multiple programs can run concurrently without exhausting physical memory.

Challenges and Solutions

- **Page fault handling**: High page fault rates can lead to high costs in terms of time and performance. Systems must efficiently manage the swap space and avoid excessive paging.
- **Instruction restarts**: When a page fault occurs during the execution of an instruction, the instruction needs to be restarted, which can add overhead. This is particularly tricky when an instruction operates on multiple memory locations or involves overlapping memory regions.

4) Segmentation

Segmentation

• This is a memory-management scheme that supports user-view of memory (Figure 1).
• A logical-address space is a collection of segments.
• Each segment has a name and a length.
• The addresses specify both segment-name and offset within the segment.
• Normally, the user-program is compiled, and the compiler automatically constructs segments reflecting the input program.
• For ex: The code, Global variables, The heap, from which memory is allocated, The stacks used by each thread, The standard C library

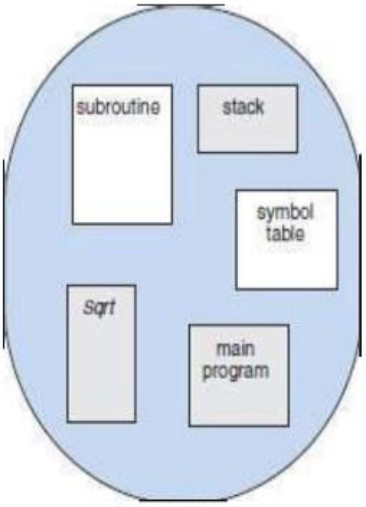


Figure: Programmer's view of a program

Hardware support for Segmentation

• Segment-table maps 2-dimensional user-defined addresses into one-dimensional physical addresses.
• In the segment-table, each entry has following 2 fields:
1. Segment-base contains starting physical-address where the segment resides my memory.
2. Segment-limit specifies the length of the segment
• A logical-address consists of 2 parts:
1. Segment-number(s) is used as an index to the segment-table
2. Offset(d) must be between O and the segment-limit.
• If the offset is not between O & segment-limit, then we trap to the OS(logical addressing attempt beyond end of segment).
• If offset is legal, then it is added to the segment-base to produce the physical-memory address.

---

5) Structure of a page table.

1. Hierarchical Paging (Two-Level Paging)

Problem:

Most systems support large logical address spaces (e.g., 32-bit or 64-bit), which results in a **large page table**. The sheer size of the page table can be inefficient in terms of memory usage.

Solution:

To tackle this, the **page table itself is divided** into smaller pieces, creating a multi-level page table structure.

- **Two-Level Paging**: The logical address is divided into multiple parts: the first part is used to index into an outer page table, and the second part is used to index into an inner page table (which contains the physical page frame number).

Example:

Consider a system with a **32-bit logical address space** and a **4 KB page size**. A logical address is divided into:

- 20-bit **page number** (for indexing the page table)
- 12-bit **offset** (for addressing within the page)

The **20-bit page number** is then divided further into:

- 10-bit **outer page-table index** (for the first-level page table)

- 10-bit **inner page-table index** (for the second-level page table)

Thus, the logical address is structured as follows:

- **Outer page table index** (p1) and **inner page table index** (p2), where p1 points to the outer page table, and p2 points to a page in the inner page table.

Address Translation:

1. The **outer page table** is accessed using the first part of the page number (p1).
2. Then, the **inner page table** is accessed using the second part of the page number (p2).
3. Finally, the **physical address** is formed by combining the frame from the inner page table with the offset.

This **two-level** method allows for smaller, more manageable page tables and reduces memory overhead.

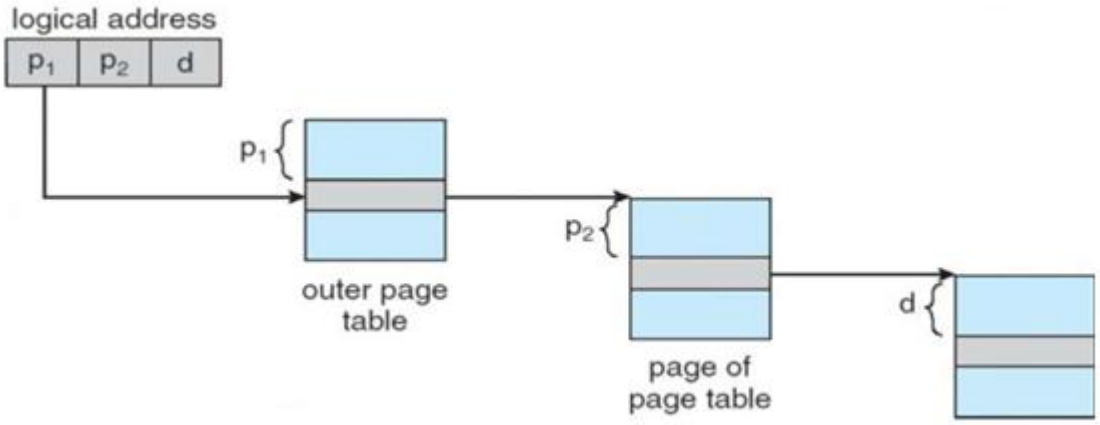
Visualization:

plaintext

Copy code

Logical Address: | 10 bits | 10 bits | 12 bits |

| Page 1 | Page 2 | Offset |



2. Hashed Page Tables

Problem:

When dealing with **large address spaces** (more than 32 bits), handling page tables can be cumbersome, especially when multiple virtual page numbers map to the same physical page number.

Solution:

A **hash table** is used to store the mapping from **virtual page numbers** to **physical page numbers**.

- **Hashing**: The **virtual page number** is hashed to find its corresponding entry in the hash table. If multiple virtual page numbers hash to the same entry, a **linked list** is used to handle collisions.

Hash Table Structure:

Each entry in the hash table contains a **linked list** with the following:

- Virtual page number
- Mapped page frame
- Pointer to the next element in the list (for handling collisions)

Algorithm:

1. The **virtual page number** is hashed to get an index in the hash table.
2. The hash table is searched for a matching **virtual page number**.
3. If a match is found, the corresponding **physical address** is retrieved using the **mapped page frame** and the **offset**.
4. If no match is found, the next entry in the linked list is checked until a match is found or the list is exhausted.

Advantages:

- Useful for systems with **larger address spaces** (greater than 32 bits).
- Efficient in handling address collisions with the linked list.

Disadvantages:

- May result in **slower lookup times** due to linked list traversal.

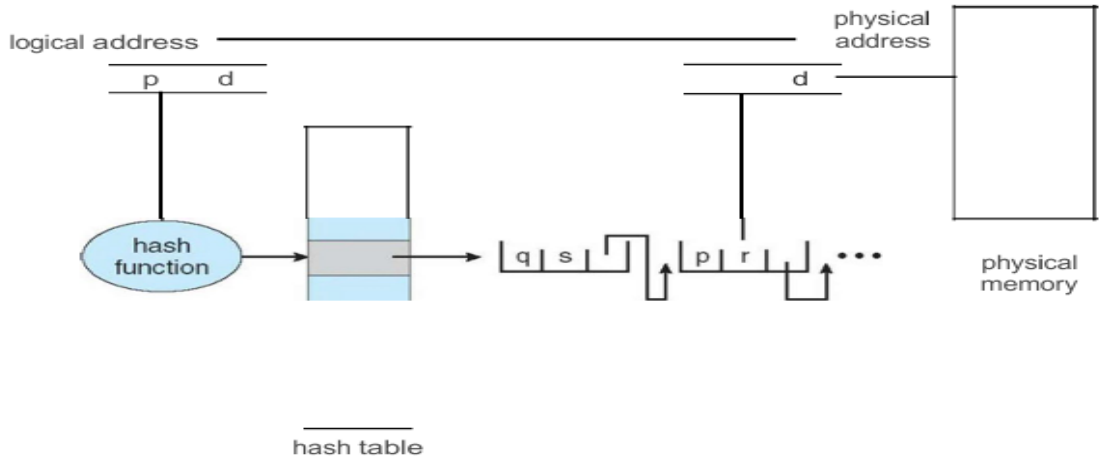
Visualization:

plaintext

Copy code

Hash Table -> Linked List (for collision handling)

|---> (Virtual page number, Frame number, Next pointer)



3. Inverted Page Tables

Problem:

Traditional page tables have one entry for each **virtual page**, but this can lead to inefficient memory usage, especially with large address spaces.

Solution:

The **inverted page table** uses **one entry per physical page** rather than one entry per virtual page. Each entry contains:

- The **virtual address** of the page stored in the corresponding physical page.
- Information about the **process** that owns the page.

Structure:

- Each entry in the inverted page table consists of a **pair**: <process-id, page-number>.
- The **virtual address** is structured as a **triplet**: <process-id, page-number, offset>, where:
  - process-id identifies which process owns the page.
  - page-number corresponds to the page number in the virtual address space.

Algorithm:

1. When a memory reference occurs, part of the **virtual address** (comprising the **process ID** and **page number**) is presented to the memory subsystem.
2. The **inverted page table** is searched for a match.
3. If a match is found, the **physical address** is generated by combining the **entry index** (i.e., physical page number) with the **offset**.
4. If no match is found, an **illegal address** has been referenced.

Advantages:

- Reduces memory usage since the table size is based on **physical pages** (smaller table size).
- Efficient for systems with **many processes**.

Disadvantages:

- Increases the **search time** for address translation because the inverted page table needs to be searched for a match.

- **Difficulty in implementing shared memory** as the mapping must account for multiple processes using the same physical pages.

Visualization:
plaintext
Copy code
Inverted Page Table:
Physical Page 0 -> (Process 1, Page 3)
Physical Page 1 -> (Process 2, Page 5)
...

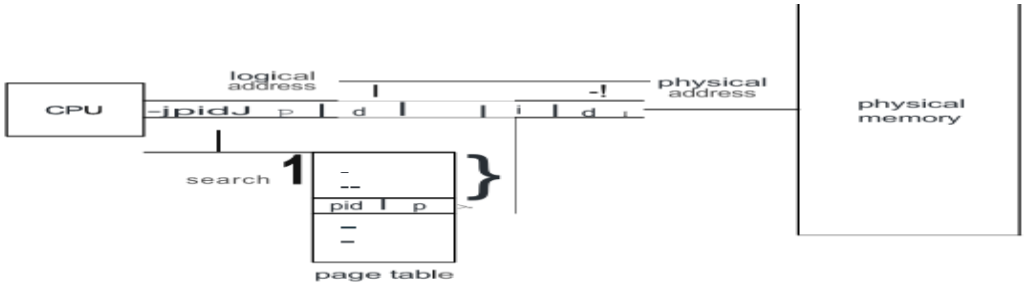


Figure: Inverted page-table

---

6) Paging and fragmentation.

Paging
• Paging is a memory-management scheme.
• This permits the physical-address space of a process to be non-contiguous.
• This also solves the considerable problem of fitting memory-chunks of varying sizes onto the backing-store.
• Traditionally: Support for paging has been handled by hardware.
• Recent designs: The hardware & OS are closely integrated.

Two types of memory fragmentation:
1. Internal fragmentation
2. External fragmentation
1. Internal Fragmentation
• The general approach is to break the physical-memory into fixed-sized blocks and allocate memory in units based on block size.
• The allocated-memory to a process may be slightly larger than the requested-memory.
• The difference between requested-memory and allocated-memory is called internal fragmentation i.e. Unused memory that is internal to a partition.
2. External Fragmentation
• External fragmentation occurs when there is enough total memory-space to satisfy a request but the available-spaces are not contiguous. (i.e. storage is fragmented into a large number of small holes).
• Both the first-fit and best-fit strategies for memory-allocation suffer from external fragmentation.
• Statistical analysis of first-fit reveals that given N allocated blocks, another 0.5 N blocks will be lost to fragmentation. This property is known as the 50-percent rule.

MODULE 5

1) File

FILE:
• A file is a named collection of related information that is recorded on secondary storage.
• The information in a file is defined by its creator. Many different types of information may be stored in a file source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on.

File Attributes:
• Name: The symbolic file name is the only information kept in human readable form.
• Identifier: This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
• Type: This information is needed for systems that support different types of files.
• Location: This information is a pointer to a device and to the location of the file on that device.
• Size: The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
• Protection: Access-control information determines who can do reading, writing, executing, and so on.
• 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.

File Operations:
1. **Creating a file:**Two steps are necessary to create a file,
a) Space in the file system must be found for the file.
b) An entry for the new file must be made in the directory.
2. **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. Given the name of the file, the systemsearches the directory to find the file's location. The system must keep a write pointer tothe location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
3. **Reading a file**:To read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per process current file-position pointer.
4. **Repositioning within a file:**The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual 1/0. This file operation is also known as files seek.
5. **Deleting a file:**To delete a file, search the directory for the named file. Having found the associated directory entry, then release all file space, so that it can be reused by other files, and erase the directory entry.
6. **Truncating a file:**The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged but lets the file be reset to length zero and its file space released

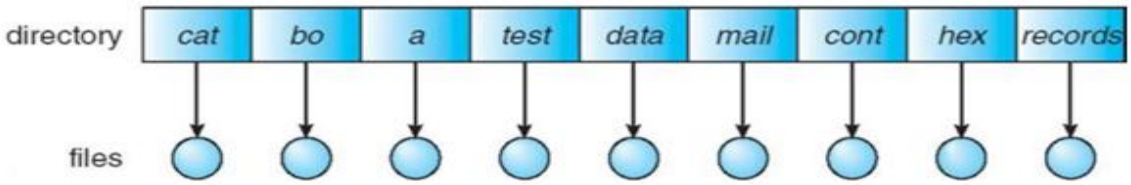
2) Directory Structures.

The most common schemes for defining the logical structure of a directory are described below

1. Single-level Directory
2. Two-Level Directory
3. Tree-Structured Directories
4. Acyclic-Graph Directories
5. General Graph Directories

1. Single-Level Directory

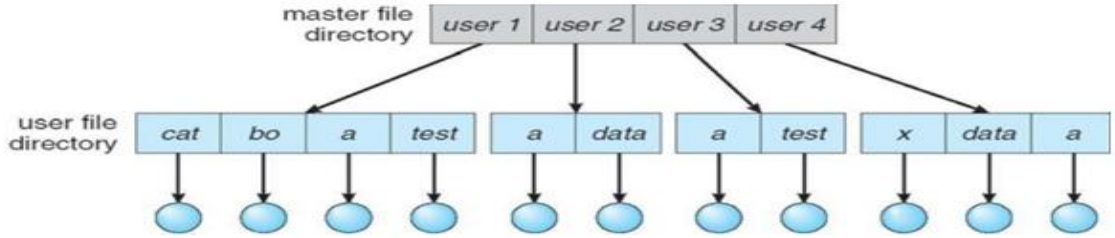
- **Description**:
  - The simplest structure, where all files are stored in one directory.
  - Every file is directly listed in the directory.
  - It's straightforward to implement and understand.
- **Problems**:
  - **File Name Uniqueness**: Since all files are in a single directory, the **file names must be unique**, which becomes difficult as the number of files or users increases.
  - **File Organization**: With many files, keeping track of files can become a difficult task, especially for a single user or in a multi-user environment.
- **Limitations**:
  - Not scalable for a large number of files or users.
  - Users may find it hard to manage files with a single directory.



---

2. Two-Level Directory

- **Description**:
  - Each user has their own **User File Directory (UFD)**, and all files for a particular user are listed in this directory.
  - The **Master File Directory (MFD)** keeps track of the UFDs and directs the operating system to the correct UFD based on the user's name or account number.
- **Advantages**:
  - **File Name Uniqueness**: No collision of file names across different users.
  - Efficient searching within a user's files, as the search is confined to the user's UFD.
- **Disadvantages**:
  - Users are isolated and cannot easily share files or collaborate on tasks in the same directory.
- **How It Works**:
  - **MFD** points to each user's **UFD**.
  - When a file is created or deleted, the operating system only searches the corresponding user's UFD.

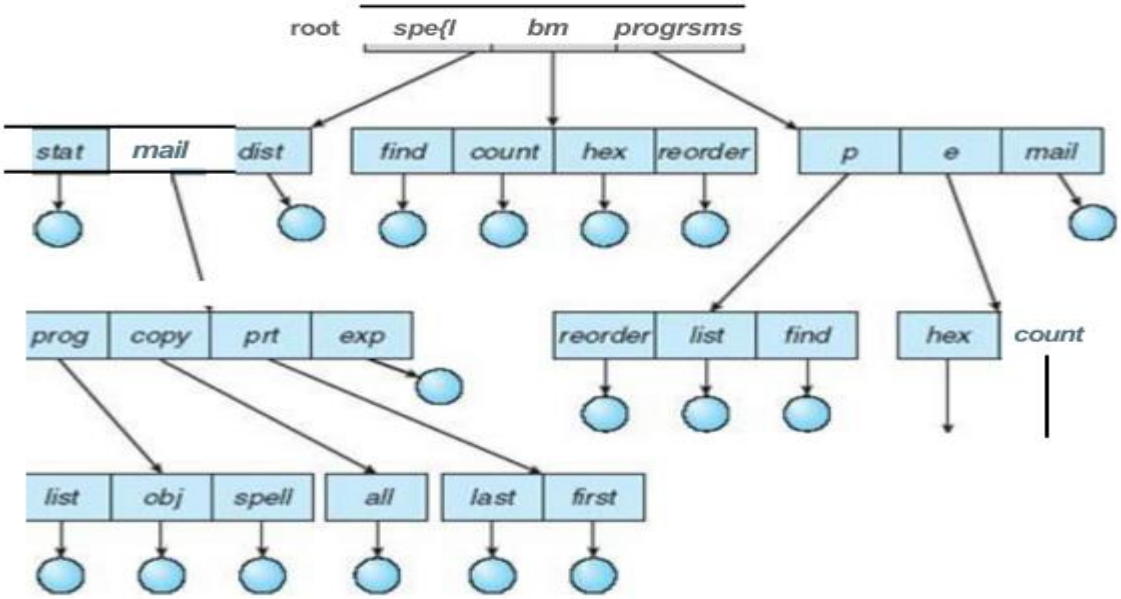


3. Tree-Structured Directories

- **Description**:
  - A hierarchical directory system, often visualized as a tree.
  - The **root directory** serves as the starting point, and each file has a **unique path**.
  - Each directory contains files or other subdirectories. Directories are treated as special files that can contain references to other directories or files.
- **Path Names**:
  - **Absolute Path Name**: A full path starting from the root.
  - **Relative Path Name**: A path relative to the current working directory.
- **How to Delete a Directory**:

1. **Empty Directory**: Simply remove the directory.
2. **Non-Empty Directory**: First delete all files and subdirectories recursively before removing the directory itself.

- **Advantages**:
  - Allows users to access the files of other users if necessary.
- **Disadvantages**:
  - Path names can be long, especially in deeply nested structures.
  - File or directory sharing is not allowed across different users unless explicitly managed.



4. Acyclic Graph Directories

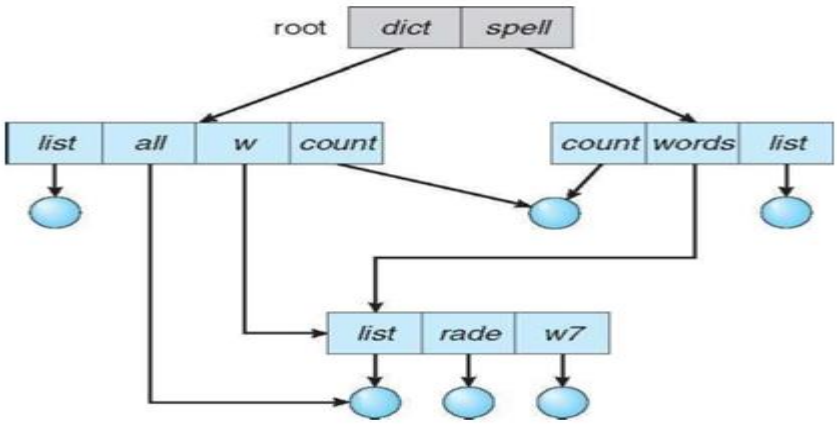
- **Description**:
  - A generalized version of tree-structured directories where directories can share files or subdirectories.
  - The structure is **acyclic**, meaning no cycles are allowed, but files or directories can exist in multiple locations.
- **How It Works**:
  - A **link** (or pointer) is used to reference a file or directory from multiple places in the file system.
  - **Two methods** to implement shared files:
    1. **Linking**: A new directory-entry (link) points to the shared file or directory.

2. **Duplication**: All information about the shared file is duplicated across different directories.

- **Problems**:

1. **Multiple Absolute Path Names**: A file may have multiple paths to access it, which can lead to confusion.
2. **Dangling Pointers**: Deletion of a file or directory may leave links to non-existent locations.

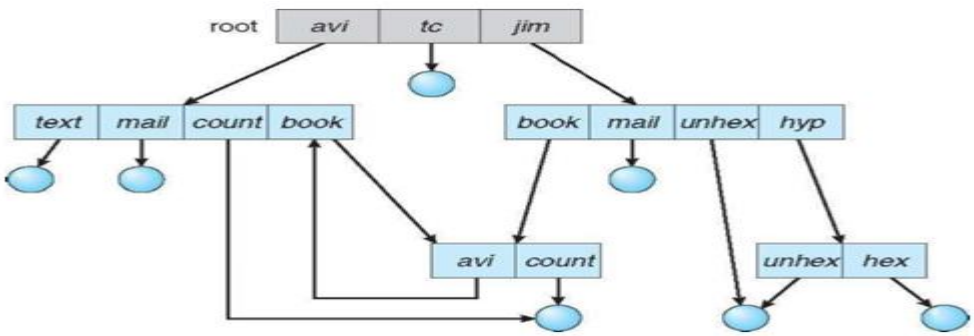
- **Solution to Deletion Problem**:
  - **Back-Pointers**: Ensure that a file is not deleted until all references (links) are deleted.
  - **Symbolic Links**: Delete only the link, not the file itself. If the file is deleted, the link can be removed safely.



---

5. General Graph Directory

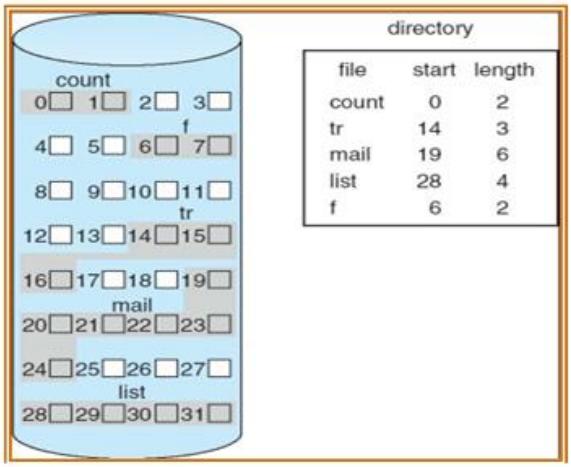
- **Description**:
  - A more generalized version of the acyclic graph, where cycles are allowed.
  - This structure requires special management to prevent **duplicate searches** and **incorrect deletions** due to cycles.
- **Problems**:
  - **Search Efficiency**: Cycles may cause components to be searched more than once.
  - **Reference Count**: If there are cycles, the reference count may not reach zero even if the file or directory is no longer accessible.
- **Solution**:
  - **Garbage Collection**:
    1. **First Pass**: Traverse the entire system and mark all accessible files and directories.
    2. **Second Pass**: Collect and remove any files or directories that are not marked as accessible, ensuring that cycles do not affect the collection process.



3) Allocation methods.

1. Contiguous Allocation:

- **Description**:
  - Files are stored in contiguous blocks on the disk.
  - The directory entry for a file contains the starting block and the length (in blocks) of the file.
  - Accessing a file is straightforward, as you know the starting block and length.
  - Supports **both sequential and direct access** to file data.
- **Advantages**:
  - **Simple Access**: For sequential access, the system keeps track of the last accessed block. For random access, it can directly compute the location of any block.
  - **Fast Access**: Since all blocks are contiguous, accessing the next block is efficient.
- **Disadvantages**:
  - **Fragmentation**: External fragmentation occurs as files are allocated and deleted, leaving gaps of free space scattered across the disk. This can make it difficult to find large enough contiguous blocks for new files.
  - **Finding Space**: Finding contiguous free space for a new file is difficult. Common strategies for allocating space (such as **First Fit** or **Best Fit**) may not always work efficiently.
  - **Inefficient Use of Space**: Over time, fragmentation may result in wasted space, making it hard to store larger files, even though there might be enough free space in total.

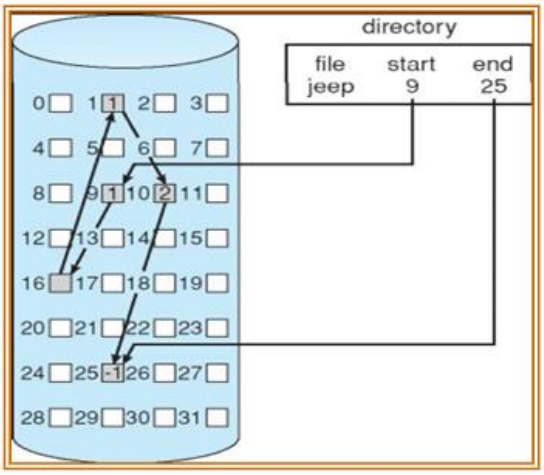


---

2. Linked Allocation:

- **Description**:
  - In this method, each file is a linked list of disk blocks. These blocks are scattered throughout the disk, and each block contains a pointer to the next block in the file.
  - The directory entry for each file contains a pointer to the first block in the file.
  - There is no need to know the size of the file in advance, and files can grow as long as there are free blocks available.
- **Advantages**:
  - **No External Fragmentation**: Files do not need contiguous space. Free blocks from anywhere on the disk can be allocated to a file.
  - **No Need to Compact**: Disk space does not need to be compacted because of fragmentation.
  - **Flexible File Size**: Files can grow without needing to pre-allocate contiguous blocks.

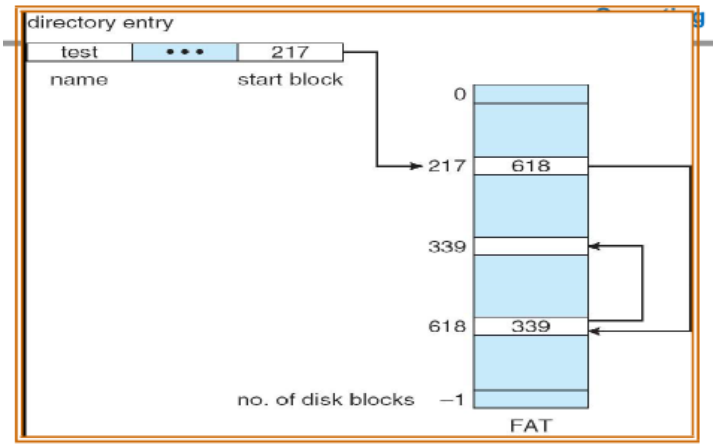
- **Disadvantages**:
    - **Sequential Access Only**: Random access is slow because to reach the i-th block, you have to follow all the pointers from the first block.
    - **Pointer Overhead**: Each block requires extra space for the pointer, reducing effective storage capacity.
    - **Reliability Issues**: If a pointer gets lost or corrupted, access to the entire file may be lost.



---

3. File Allocation Table (FAT):

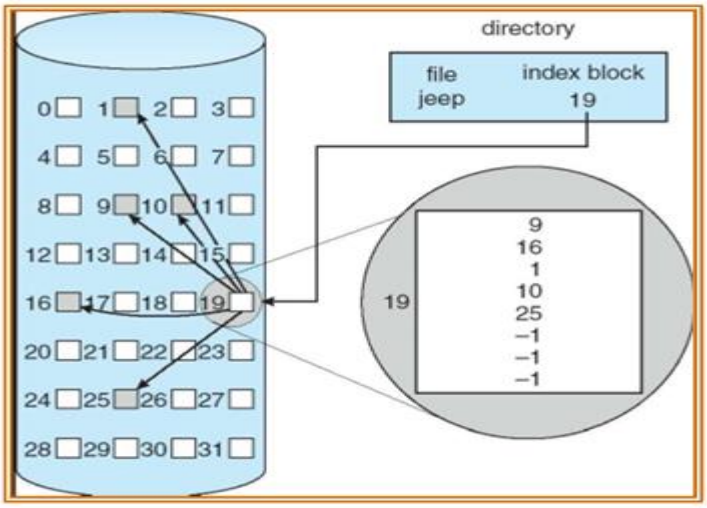
- **Description**:
    - A section of the disk is reserved as the **File Allocation Table (FAT)**, which is an array of entries, one for each block on the disk.
    - The FAT entry for a block contains the block number of the next block in the file. The directory entry contains the block number of the first block of the file.
    - The chain continues until the last block, which contains a special end-of-file (EOF) marker.
- **Advantages**:
    - **Efficient File Access**: Similar to linked allocation, but the FAT allows quicker access to the next block of the file since the FAT is stored in a contiguous location, making it faster to access the file's chain of blocks.
- **Disadvantages**:
    - **Table Size**: The FAT table must be loaded into memory, and for large disks with many blocks, the table can become large, leading to higher memory and I/O overhead.



---

4. Indexed Allocation:

- **Description**:
  - Instead of having pointers in each data block, **indexed allocation** stores the pointers in a special index block.
  - Each file has an **index block**, which is an array of pointers to the actual data blocks.
  - The directory entry contains the address of the index block.
- **Advantages**:
  - **Direct Access**: Supports direct access to any block in the file by using the index block, which makes it faster for random access than linked allocation.
  - **No External Fragmentation**: Like linked allocation, indexed allocation does not require contiguous space.
  - **Efficient Space Use**: The index block stores all the pointers in one place, avoiding fragmentation of the file's data blocks.
- **Disadvantages**:
  - **Pointer Overhead**: The index block requires space for storing pointers. This can be inefficient if the file is small, and there is a waste of space if the index block is not fully utilized.
  - **Large Files**: If the file grows large enough to exceed the space available in one index block, a multi-level indexing system may be required (which can add complexity).



---

4) Access methods.

1. Sequential Access Method:

- **Description**: The simplest access method where information in the file is processed in sequence, one record after the other.
- **Operations**:
  - **Read Operation (next-read)**: Reads the next portion of the file and automatically moves the file pointer.
  - **Write Operation (write-next)**: Appends to the end of the file and moves to the end of the newly written material.
  - **Reset**: The file pointer can be reset to the beginning of the file, or sometimes skip forward/backward by a set number of records.

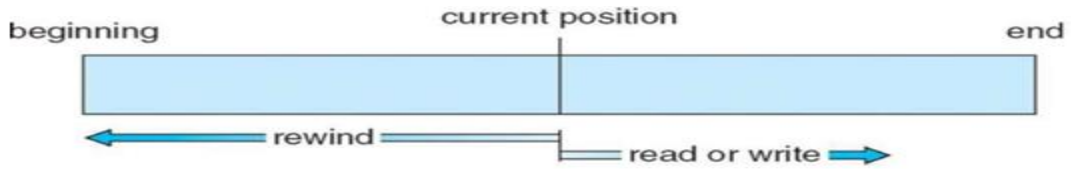


Figure: Sequential-access file.

---

2. Direct Access Method:

- **Description**: Files are made up of fixed-length logical records that allow access in no particular order. The file is viewed as a numbered sequence of blocks or records.
- **Operations**:
  - **Read Operation**: Access any block (e.g., read block 14, then block 53).
  - **Write Operation**: Write to any block (e.g., write block 7).
  - **File Positioning**: Instead of reading or writing sequentially, you specify the block number (e.g., position to block 7, then read or write).

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

---

3. Other Access Methods:

- **Description**: Methods built on top of direct access, often involving an index.
  - **Indexing**: Similar to a book index, it stores pointers to various blocks, allowing quick access to specific records. To find a record, search the index and use the pointer to access the file directly.

---

5) Protection domain.

PROTECTION

- When information is stored in a computer system, we want to keep it safe from physical damage (reliability) and improper access (protection).
- Reliability is generally provided by duplicate copies of files.
- For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer.
- File owner/creator should be able to control what can be done and by whom.

Types of Access

- Systems that do not permit access to the files of other users do not need protection. This is too extreme, so controlled-access is needed.

Following operations may be controlled:

1. Read: Read from the file.
2. Write: Write or rewrite the file.
3. Execute: Load the file into memory and execute it.
4. Append: Write new information at the end of the file.
5. Delete: Delete the file and tree its space for possible reuse.

6. List: List the name and attributes of the file.

Access Control

• Common approach to protection problem is to make access dependent on identity of user.

• Files can be associated with an ACL (access-control list) which specifies usemame andtypes of access for each user.

Problems:

1. Constructing a list can be tedious.

2. Directory-entry now needs to be of variable-size, resulting in more complicated space Management

Solution:

• These problems can be resolved by combining ACLs with an 'owner, group, universe' access control scheme

• To reduce the length of the ACL, many systems recognize 3 classifications of users:

1. Owner: The user who created the file is the owner.

2. Group: A set of users who are sharing the file and need similar access is a group.

3. Universe: All other users in the system constitute the universe.

---

6) File types

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	*ps, pdf, jpg*	*ASCII or binary file in a* format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

---

7) Access Matrix.

- The **access matrix** is a model for implementing protection mechanisms without imposing a specific policy.
- **Rows** represent domains (execution environments or processes).
- **Columns** represent objects (files, printers, etc.).
- Each entry in the matrix specifies the **set of access rights** a domain has over an object.

Key Features

- **Entry Access(i, j)**: Specifies the operations a process in domain DiD_iDi can perform on object OjO_jOj.
- **Domain Switching**: Processes can switch domains (e.g., D2D_2D2 to D3D_3D3) if explicitly allowed by the access matrix.
- **Rights Propagation**:
    - **Copy Right (R*)**: Enables copying rights within the same column. Variants include:
        1. **Transfer Right**: Moves a right from one domain to another.
        2. **Limited Copy**: Creates a non-propagable copy (R).
    - **Owner Right**: Allows adding or removing rights within a column.
    - **Control Right**: Allows removing rights within a row.

Implementation Methods

The sparse nature of the access matrix necessitates efficient implementations:

1. **Global Table**
    - Represents the matrix as a list of ordered triples <domain, object, rights-set><\text{domain, object, rights-set}><domain, object, rights-set>.
    - Operations involve searching the table for the corresponding triple.
    - **Drawback**: Large size and additional I/O requirements make this method inefficient.
2. **Access Lists for Objects**
    - Each object (column) maintains a list of ordered pairs <domain, rights-set><\text{domain, rights-set}><domain, rights-set>.
    - If a domain tries to perform an operation, the list is checked for permissions.
    - A **default set** of rights can be defined for unlisted domains.
3. **Capability Lists for Domains**
    - Each domain maintains a list of objects and associated access rights.
    - **Capability**: A token representing the allowed operations on an object.
    - **Protection Mechanisms**:
        1. Tags to distinguish capabilities from other data.
        2. Restricted address space for capabilities accessible only by the OS.
4. **Lock-Key Mechanism**
    - Objects have a list of unique **locks**, and domains have a list of **keys**.
    - Access is granted if a domain's key matches an object's lock.
    - A balanced approach combining features of access and capability lists.

8) Free Space lists

a) Bit Vector

- **Description**: A bit vector represents the status of each disk block, with 1 indicating a free block and 0 indicating an allocated block.
- **Example**:
  For a disk with blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, and 18 free, the bit vector is:
  0011110011111100011
- **Advantages**:
  - Efficient algorithms for finding the first free block or contiguous blocks.
  - Easy to implement.
- **Disadvantages**:
  - Requires significant storage space (e.g., a 40GB disk needs 5MB for the bitmap).

---

b) Linked List

- **Description**: Free blocks are organized as a linked list. Each free block contains a pointer to the next free block.
- **Advantages**:
  - Simple to implement.
  - Suitable for dynamic addition and removal of blocks.
- **Disadvantages**:
  - Traversing the list is inefficient, especially when searching for contiguous free blocks.
  - Operations to find specific blocks are slow.
- **Usage**: The FAT file system includes free space tracking as a linked list in its table.

---

c) Grouping

- **Description**: Enhances the linked list approach by storing addresses of multiple free blocks (e.g., nnn) in the first block. The nnn-th block stores the addresses of the next group of free blocks.
- **Advantages**:
  - Allows quick access to many free block addresses.
  - Reduces traversal overhead compared to a simple linked list.
- **Disadvantages**:
  - Slightly more complex than a basic linked list.

---

d) Counting

- **Description**: Tracks contiguous free blocks using the starting address and the count of blocks.
- **Advantages**:
  - Reduces storage requirements by avoiding individual addresses for each block.
  - Efficient for managing contiguous free blocks.
  - Similar to the extent-based block allocation method.
- **Disadvantages**:
  - Less efficient for fragmented free space.

---

e) Space Maps

- **Description**:
  - Utilized by Sun's ZFS for efficient management of large disks and files.
  - Divides the disk into **Meta-slabs**, each with its own space map.

    - Uses a **counting technique** to manage free blocks, storing the data in a log-structured format rather than a table.
    - Free blocks are coalesced into larger blocks.
    - An **in-memory balanced tree** represents the space map for fast lookup and updates.
- **Advantages**:
    - Highly efficient for managing large and complex file systems.
    - Reduces overhead for operations involving large files.
- **Disadvantages**:
    - More complex implementation compared to other methods.
    - Requires more memory for maintaining in-memory structures.
