Unit-4

File Management, Deadlock and File Security Techniques

Computer users store programs and data in files so that they can be used conveniently and preserved across computing sessions. A user has many expectations when working with files, namely

- Convenient and fast access to files
- Reliable storage of files
- Sharing of files with collaborators

The resources used for storing and accessing files are I/O devices. As it must, the OS ensures both efficient performance of file processing activities in processes and efficient use of I/O devices.

Operating systems organize file management into two components called the *file system* and the *input-output control system* (IOCS) to separate the file-level concerns from concerns related to efficient storage and access of data. Accordingly, a file system provides facilities for creating and manipulating files, for ensuring reliability of files when faults such as power outages or I/O device malfunctions occur, and for specifying how files are to be shared among users. The IOCS provides access to data stored on I/O devices and good performance of I/O devices.

➢ Overview of File Processing

The term file processing to describe the general sequence of operations of opening a file, reading data from the file or writing data into it, and closing the file. Figure 13.1 shows the arrangement through which an OS implements file processing activities of processes. Each directory contains entries describing some files. The directory entry of a file indicates the name of its owner, its location on a disk, the way its data is organized, and which users may access it in what manner. The code of a process Pi is shown in the left part of Figure 13.1. When it opens a file for processing, the file system locates the file through the directory structure, which is an arrangement of many directories. In Figure 13.1, there are two files named beta located in different directories. When process Pi opens beta, the manner in which it names beta, the directory structure, and identities of the user who initiated process Pi will together determine which of the two files will be accessed.

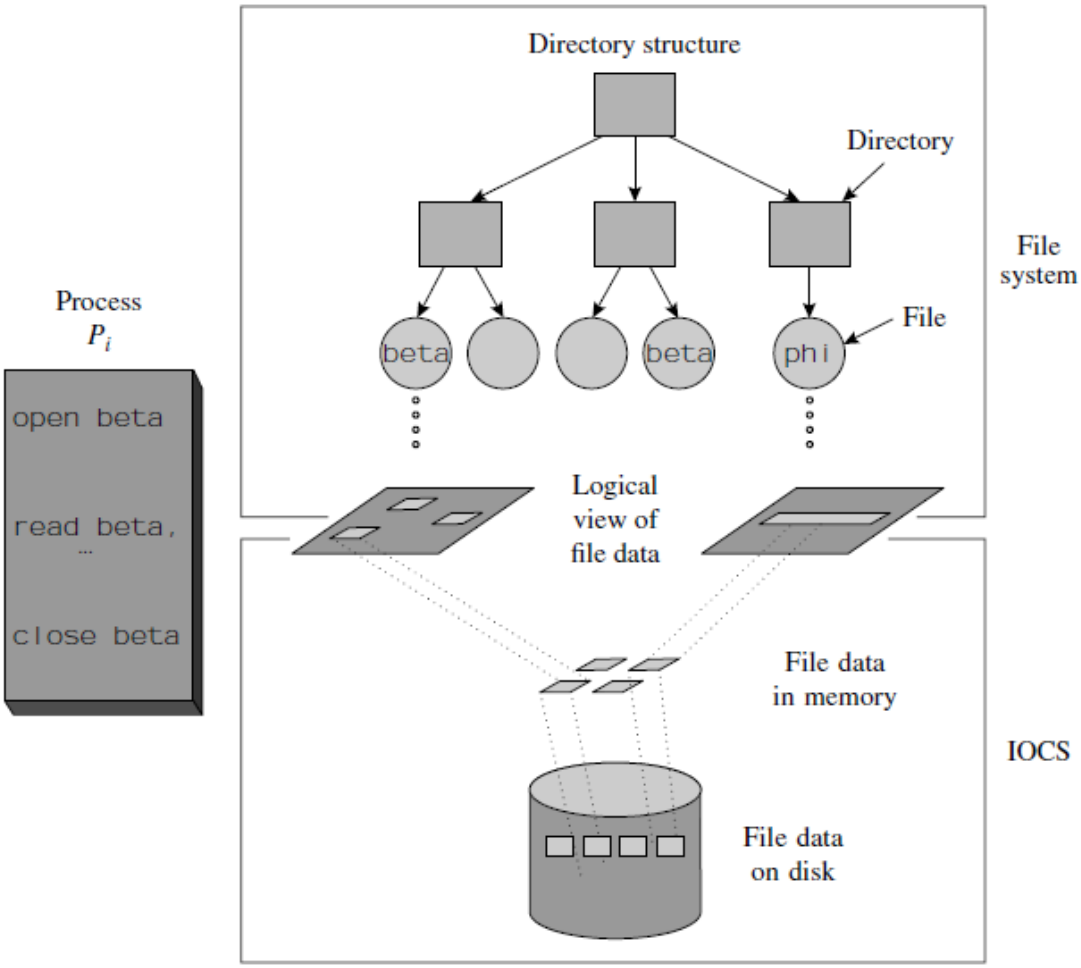


Figure 13.1 File system and the IOCS.

A file system provides several file types (see Section 13.2). Each file type provides its own abstract view of data in a file—we call it a logical view of data. Figure 13.1 shows that file beta opened by process Pi has a record-oriented logical view, while file phi has a byte stream–oriented logical view in which distinct records do not exist.

The IOCS organizes a file's data on an I/O device in accordance with its file type. It is the physical view of the file's data. The mapping between the logical view of the file's data and its physical view is performed by the IOCS. The IOCS also provides an arrangement that speeds up a file processing activity—it holds some data from a file in memory areas organized as buffers, a file cache, or a disk cache. When a process performs a read operation to get some data from a file, the IOCS takes the data from a buffer or a cache if it is present there. This way, the process does not have to wait until the data is read off the I/O device that holds the file. Analogously, when a process performs a write operation on a file, the IOCS copies the data to be written in a buffer or in a cache.

➢ File System and the IOCS

Table 13.1 summarizes the facilities provided by the file system and the IOCS. The file system provides directory structures that enable users to organize their data into logical groups of files, e.g., one group of files for each professional activity. The file system provides protection against illegal file accesses and ensures correct operation when processes access and update a file concurrently. It also ensures that data is reliably stored, i.e., data is not lost when system crashes occur.

The IOCS policy modules ensure efficient operation of I/O devices and efficient file processing in each process through the IOCS mechanism modules The mechanism modules in the IOCS, in turn, invoke the kernel through system calls to initiate I/O operations.

Table 13.1 Facilities Provided by the File System and the Input-Output Control System

File System
• Directory structures for convenient grouping of files
• Protection of files against illegal accesses
• File sharing semantics for concurrent accesses to a file
• Reliable storage of files
Input-Output Control System (IOCS)
• Efficient operation of I/O devices
• Efficient access to data in a file

➢ Files and File Operations

File Types:

It refers to the ability of the operating system to differentiate various types of files e.g., data files, executable programs, object modules, textual information, documents, spreadsheets, photos, and video clips. Each of these file types has its own format for recording the data.

File Attribute:

A file has a name and data. Moreover, it also stores information like file creation date and time, current size, last modified date, etc. All this information is called the attributes of a file system.

Some important File attributes used in OS:

- **Name:** It is the only information stored in a human-readable form.
- **Identifier**: Every file is identified by a unique tag number within a file system known as an identifier.
- **Location:** Points to file location on device.
- **Size**. Attribute used to display the current file size.
- **Protection**. This attribute assigns and controls the access rights of reading, writing, and executing the file.
- **Time, date and security:** It is used for protection, security, and also used for monitoring

File Operations:

File operations such as open, close, rename, and delete are performed by file system modules.

Table 13.2 Operations on Files

Operation	Description
Opening a file	The file system finds the directory entry of the file and checks whether the user whose process is trying to open the file has the necessary access privileges for the file. It then performs some housekeeping actions to initiate processing of the file.
Reading or writing a record	The file system considers the organization of the file (see Section 13.3) and implements the read/write operation in an appropriate manner.
Closing a file	The file size information in the file's directory entry is updated.
Making a copy of a file	A copy of the file is made, a new directory entry is created for the copy and its name, size, location, and protection information is recorded in the entry.
File deletion	The directory entry of the file is deleted and the disk area occupied by it is freed.
File renaming	The new name is recorded in the directory entry of the file.
Specifying access privileges	The protection information in the file's directory entry is updated.

➢ File Processing in a Program

At the programming language level, a file is an object that possesses attributes describing the organization of its data and the method of accessing the data. A program contains a declaration statement for a file, which specifies values of its attributes, and statements that open it, perform read/write operations on it, and close it (we call them file processing statements). During execution of the program, file processing is actually implemented by library modules of the file system and the IOCS.

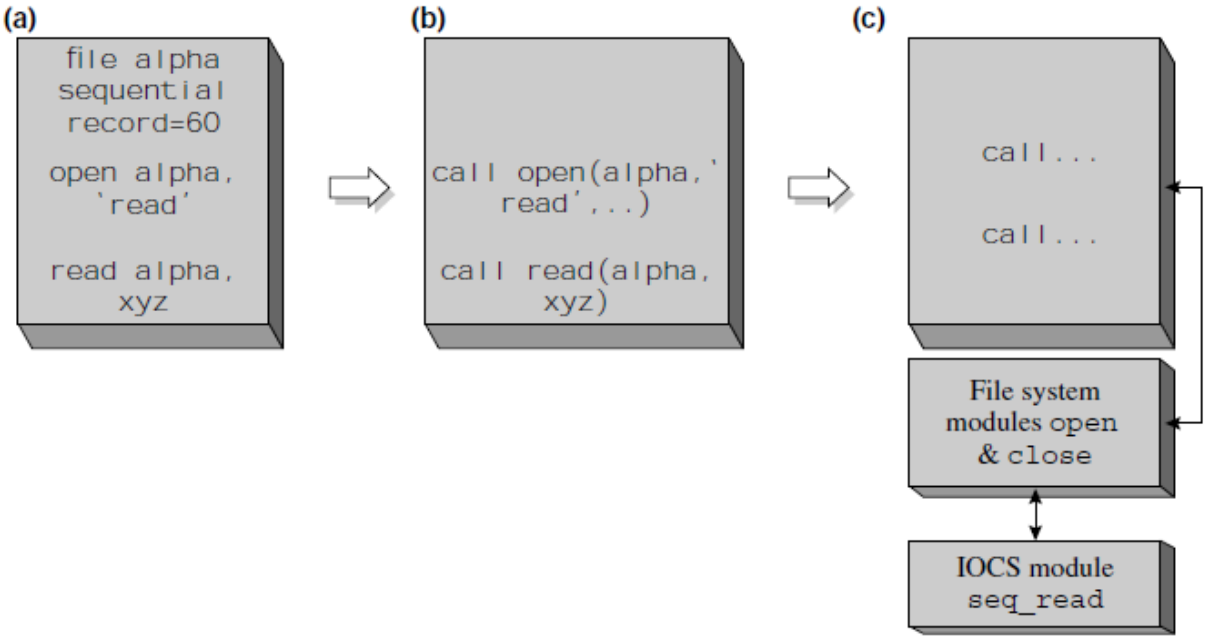


Figure 13.2 Implementing a file processing activity: (a) program containing file declaration statements; (b) compiled program showing calls on file system modules; (c) process invoking file system and IOCS modules during operation.

Figure 13.2 illustrates how file processing is actually implemented. The program of Figure 13.2(a) declares alpha as a sequential-access file that contains records with a size of 60 bytes (see Section 13.2 for a discussion of records in a file). It also contains statements to open alpha and read a record from it. The compiler of the programming language processes the file declaration statement in the program and determines attributes of the file. It now replaces open, close, read, and write statements with calls on file system library modules open, close, read, and write, and passes the file attributes as parameters to the open call [see Figure 13.2(b)]. The file system modules invoke modules of the IOCS to actually perform I/O operations. The linker links the file system library modules and the IOCS modules invoked by them to produce the program shown in Figure 13.2(c) (see Section 11.3.2 for a description of the linking function). When a process is created for execution of this program, it invokes the file system library modules during its operation to

perform the open and read operations on the file, and these modules implement them with the help of appropriate IOCS library modules.

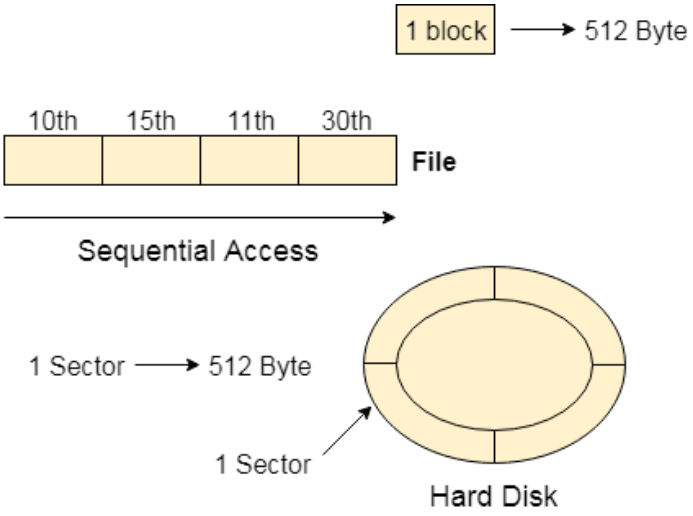
➢ File Access Methods

File access is a process that determines the way that files are accessed and read into memory. Generally, a single access method is always supported by operating systems. Though there are some operating system which also supports multiple access methods.

Three file access methods are:

- Sequential access
- Direct random access
- Index sequential access

Sequential Access:



Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.

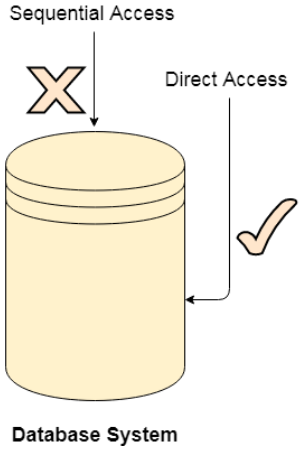
In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.

Direct Access:

The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.

Suppose every block of the storage stores 4 records and we know that the record we needed is stored in 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record.

Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.



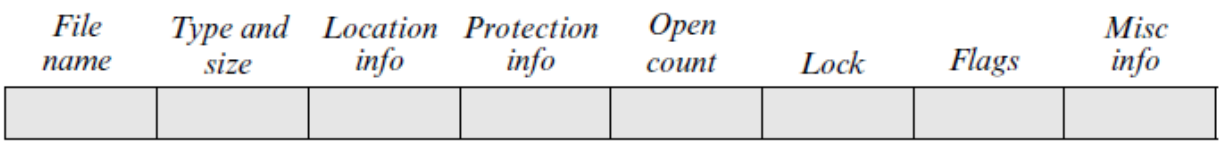
Indexed Access:

If a file can be sorted on any of the filed then an index can be assigned to a group of certain records. However, A particular record can be accessed by its index. The index is nothing but the address of a record in the file.

In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.

➢ Directories

A directory contains information about a group of files. Each entry in a directory contains the attributes of one file, such as its type, organization, size, location, and the manner in which it may be accessed by various users in the system. Figure 13.6 shows the fields of a typical directory entry.



Field	Description
File name	Name of the file. If this field has a fixed size, long file names beyond a certain length will be truncated.
Type and size	The file's type and size. In many file systems, the type of file is implicit in its extension; e.g., a file with extension .c is a byte stream file containing a C program, and a file with extension .obj is an object program file, which is often a structured file.
Location info	Information about the file's location on a disk. This information is typically in the form of a table or a linked list containing addresses of disk blocks allocated to a file.
Protection info	Information about which users are permitted to access this file, and in what manner.
Open count	Number of processes currently accessing the file.
Lock	Indicates whether a process is currently accessing the file in an exclusive manner.
Flags	Information about the nature of the file—whether the file is a directory, a link, or a mounted file system.
Misc info	Miscellaneous information like id of owner, date and time of creation, last use, and last modification.

The *open count* and lock fields are used when several processes open a file concurrently. The open *count indicate* the number of such processes. As long as this count is nonzero, the file system keeps some of the metadata concerning the file in memory to speed up accesses to the data in the file. The lock field is used when a process desires exclusive access to a file. The flags field is used to differentiate between different kinds of directory entries. We put the value "D" in this field to indicate that a file is a directory, "L" to indicate that it is a link, and "M" to indicate that it is a mounted file system. Later sections in this chapter will describe these uses. The misc info field contains information such as the file's owner, its time of creation, and last modification.

A file system houses files owned by several users. Therefore it needs to grant users two important prerogatives:

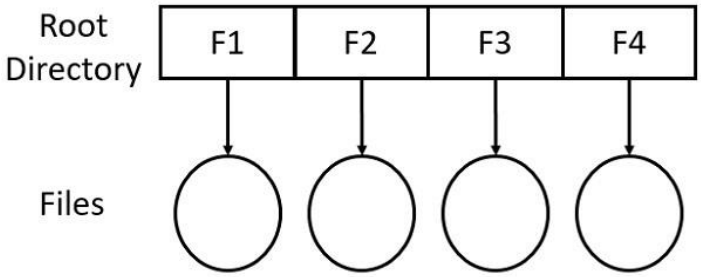
- ***File naming freedom:*** A user's ability to give any desired name to a file, without being constrained by file names chosen by other users.

• ***File sharing:*** A user's ability to access files created by other users, and ability to permit other users to access his files.

➢ Types of Directory Structures

1. Single-level directory structure

Single level directory structure has only one directory which is called the root directory. The users are not allowed to create subdirectories under the root directory. All the files created by the several users are present in the root directory only.

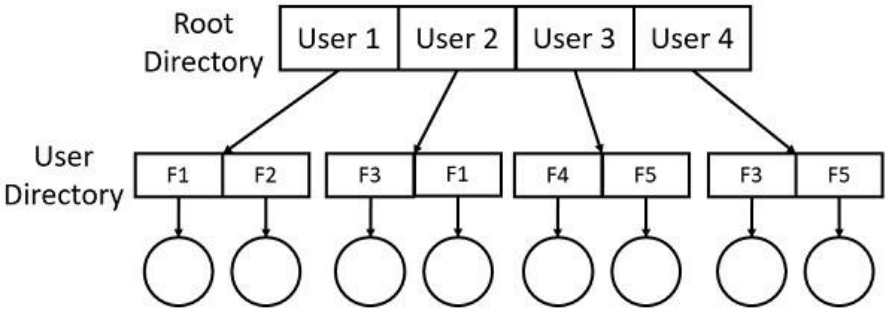


Single-level Directory Structure

There is one drawback of Single-level directory structure; a user cannot use the same file name used by another user in the system. Even if the file with the same name is created the old file will get destroyed first and replaced by the new file having the same name.

2. Two-level directory structure

In Two-level directory structure, the users create directory directly inside the root directory. But once a user creates such directory, further he cannot create any subdirectory inside that directory. In the figure below, 4 users have created their separate directory inside the root directory. But further, no subdirectory is created by the users.

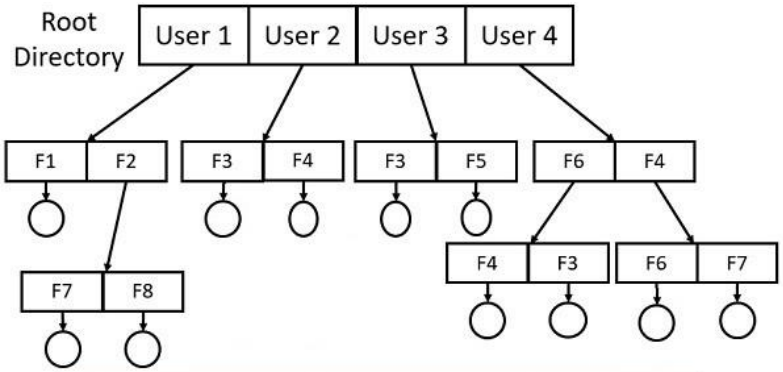


Two-level Directory Structure

This two-level structure allows each user to keep their files separately inside their own directory. This structure allows to use the same name for the files but under different user directories.

3. Hierarchical Directory Structure

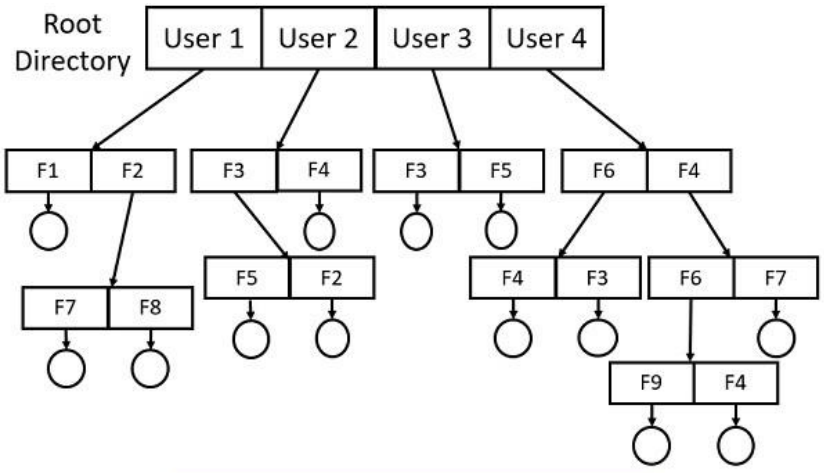
In Hierarchical directory structure, the users can create directories under the root directory and can also create sub-directories under this structure. As the user is free to create many sub-directories, it can create different sub-directories for different file types.



Hierarchical Directory Structure

4. Tree Directory Structure

In a tree directory structure, except root directory, every directory or file has only **one parent directory**. So, there is a total separation between the users which provide complete naming freedom. Here, if a user wishes to access another users file, it has to go through two or more directories.



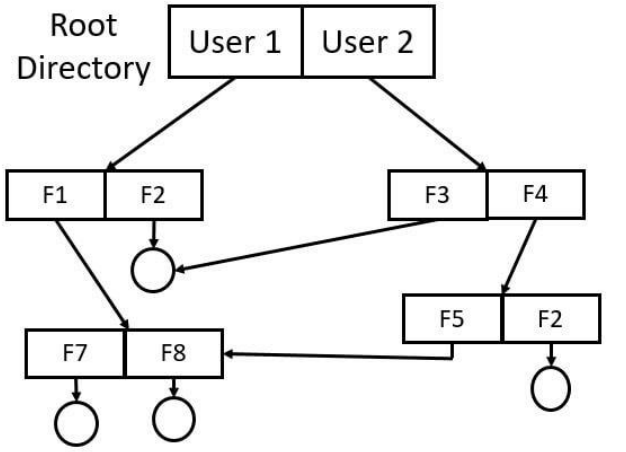
Tree Directory Structure

The tree directory structure provides an asymmetric way for the user to access the shared files of a different user. For example, a user can access a file of its own user directory with a shorter path than the other user.

5. Acyclic-Graph Directory Structure

This problem can be solved by the acyclic-graph directory structure. As this directory structure allows a directory or a file to have many parent directories. So, a shared file in a directory can be pointed by the other user directories who have access to that shared file using the links.

In the diagram below you can see that the directory having file F7 and F8 have two parent directories.



Acyclic Graph Directory Structure

➢ Operations on Directory

A directory contains the entries of all the related files. For organizing the directory in the better way the user must be able to insert, delete, search, and list the entries in the directory. The below operations that can be performed on the directory.

- **Searching**

A directory can be searched for a particular file or for another directory. It can also be searched to list all the files with the same name.

- **Creating**

A new file can be created and inserted to the directory or new directory can be created keeping in mind that its name must be unique under that particular directory.

- **Deleting**

If a file is no longer needed by the user, it can be deleted from the directory. The entire directory can also be deleted if it is not needed. An empty directory can also be deleted. When a directory is empty it is resembled by dot and dotdot.

- **List a directory**

List of all the files in the directory can be retrieved and also the contents of the directory entry, for each file in a list. To read the list of all the files in the directory, it must be opened and after reading the directory must be closed to free up the internal table space.

- **Renaming**

The name of the file or a directory represents the content it holds and its use. The file or directory can be renamed in case, the content inside or the use of file gets changed. Renaming the file or directory also changes its position inside the directory.

- **Link**

The file can be allowed to appear in more than one directory. Here, the system call creates a link between the file and the name specified by the path where the file is to appear.

- **Unlink**

If the file is unlinked and is only present in one directory its directory entry is removed. If the file appears in multiple directories, only the link is removed.

➢ Deadlocks

A *deadlock* is a situation concerning a set of processes in which each process in the set waits for an event that must be caused by another process in the set. Each process is then waiting for an event that cannot occur. Example 8.1 illustrates how a deadlock could arise when two processes try to share resources.

Example 8.1 Two-Process Deadlock

A system contains one tape drive and one printer. Two processes Pi and Pj make use of the tape drive and the printer through the following programs:

Process P_i	*Process* P_j
Request tape drive; Request printer; Use tape drive and printer; Release printer; Release tape drive;	Request printer; Request tape drive; Use tape drive and printer; Release tape drive; Release printer;

As the two processes execute, resource requests take place in the following order:

1. Process Pi requests the tape drive
2. Process Pj requests the printer
3. Process Pi requests the printer
4. Process Pj requests the tape drive

The first two resource requests are granted right away because the system includes both a tape drive and a printer. Now, Pi holds the tape drive and Pj holds the printer. When Pi asks for the printer, it is blocked until Pj releases the printer. Similarly, Pj is blocked until Pi releases the tape drive. Both processes are blocked indefinitely because they wait for each other.

➢ Deadlocks in Resource Allocation

In General, a process must request a resource before using it and it must release the resource after using it. And any process can request as many resources as it requires in order to complete its designated task. And there is a condition that the number of resources requested may not exceed the total number of resources available in the system.

Basically in the Normal mode of Operation utilization of resources by a process is in the following sequence:

1. **Request:** A process requests a resource through a system call. If the resource is free, the kernel allocates it to the process immediately; otherwise, it changes the state of the process to *blocked*.

2. **Use:** The process becomes the holder of the resource allocated to it. The resource state information is updated and the state of the process is changed to *ready*.

3. **Release:** A process releases a resource through a system call. If some processes are blocked on the allocation event for the resource, the kernel uses some tie-breaking rule, e.g., FCFS allocation, to decide which process should be allocated the resource.

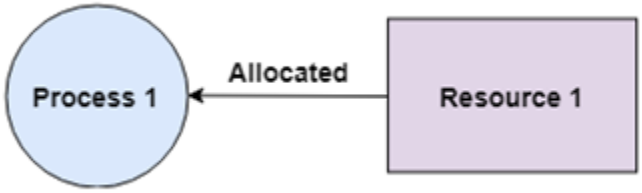
➢ Conditions for resource allocation

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

The Coffman conditions are given as follows –

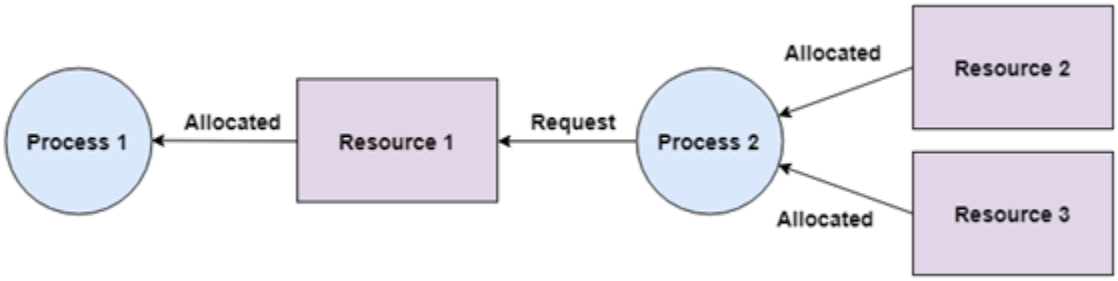
- **Mutual Exclusion/ Non-shareable resources**

  There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



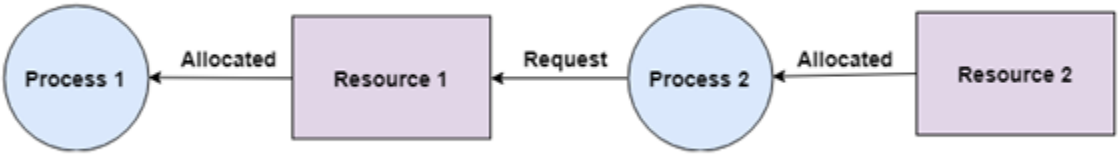
- **Hold and Wait**

  A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



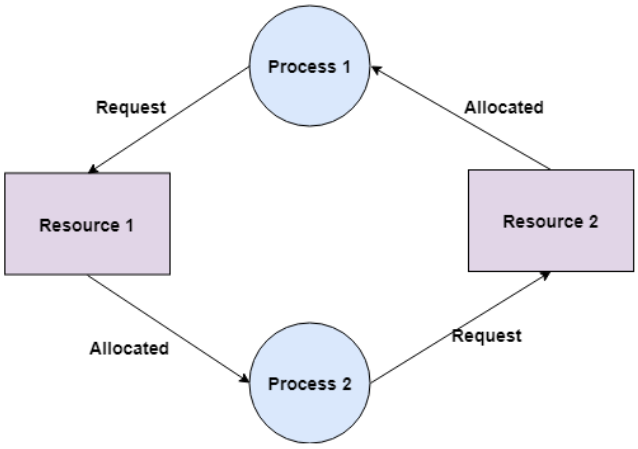
- **No Preemption**

  A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



- **Circular Wait**

  A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



➢ Handling Deadlocks

Methods that are used in order to handle the problem of deadlocks are as follows:

Deadlock Prevention:

All four conditions: Mutual Exclusion, Hold and Wait, No preemption, and circular wait if held by a system then causes deadlock to occur. The main aim of the deadlock prevention method is to violate any one condition among the four; because if any of one condition is violated then the problem of deadlock will never occur. As the idea behind this method is simple but the difficulty can occur during the physical implementation of this method in the system.

Mutual Exclusion

This condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. In contrast, Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock. A good example of a sharable resource is Read-only files because if several processes attempt to open a read-only file at the same time, then they can be granted simultaneous access to the file.

A process need not to wait for the sharable resource. Generally, deadlocks cannot be prevented by denying the mutual exclusion condition because there are some resources that are intrinsically non-sharable.

Hold and Wait

Hold and wait condition occurs when a process holds a resource and is also waiting for some other resource in order to complete its execution. Thus if we did not want the occurrence of this condition then we must guarantee that when a process requests a resource, it does not hold any other resource.

There are some protocols that can be used in order to ensure that the Hold and Wait condition never occurs:

- According to the first protocol; Each process must request and gets all its resources before the beginning of its execution.
- The second protocol allows a process to request resources only when it does not occupy any resource.

Disadvantages of Both Protocols

- Utilization of resources may be low, since resources may be allocated but unused for a long period. In the above-given example, for instance, we can release the DVD drive and disk file and again request the disk file and printer only if we can be sure that our data will remain on the disk file. Otherwise, we must request all the resources at the beginning of both protocols.
- There is a possibility of starvation. A process that needs several popular resources may have to wait indefinitely because at least one of the resources that it needs is always allocated to some other process.

No Preemption

The third necessary condition for deadlocks is that there should be no preemption of resources that have already been allocated. In order to ensure that this condition does not hold the following protocols can be used:

- According to the First Protocol: "If a process that is already holding some resources requests another resource and if the requested resources cannot be allocated to it, then it must release all the resources currently allocated to it."

- According to the Second Protocol: "When a process requests some resources, if they are available, then allocate them. If in case the requested resource is not available then we will check whether it is being used or is allocated to some other process waiting for other resources. If that resource is not being used, then the operating system preempts it from the waiting process and allocate it to the requesting process. And if that resource is being used, then the requesting process must wait".

The second protocol can be applied to those resources whose state can be easily saved and restored later for example CPU registers and memory space, and cannot be applied to resources like printers and tape drivers.

Circular Wait

The Fourth necessary condition to cause deadlock is circular wait, In order to ensure violate this condition: Assign a priority number to each resource. There will be a condition that any process cannot request for a lesser priority resource. This method ensures that not a single process can request a resource that is being utilized by any other process and due to which no cycle will be formed.

S.No	Necessary Conditions	Approach	Practical Implementation
1	Mutual Exclusion	The approach used to violate this condition is spooling.	Not possible
2	Hold and wait	In order to violate this condition, the approach is to request all the resources for a process initially	Not possible
3	No Preemption	In order to violate this condition, the approach is: snatch all the resources from the process.	Not possible
4	Circular Wait	In this approach is to assign priority to each resource and order them numerically	**possible**

Deadlock Avoidance in Operating System

The deadlock Avoidance method is used by the operating system in order to check whether the system is in a safe state or in an unsafe state and in order to avoid the deadlocks, the process must need to tell the operating system about the maximum number of resources a process can request in order to complete its execution.

In this method, the request for any resource will be granted only if the resulting state of the system doesn't cause any deadlock in the system. This method checks every step performed by the operating system. Any process continues its execution until the system is in a safe state. Once the system enters into an unsafe state, the operating system has to take a step back.

Deadlock Avoidance Example

Consider a system having 12 magnetic tapes and three processes P1, P2, P3. Process P1 requires 10 magnetic tapes, process P2 may need as many as 4 tapes, and process P3 may need up to 9 tapes. Suppose at a time t_o, process P1 is holding 5 tapes, process P2 is holding 2 tapes and process P3 is holding 2 tapes. (There are 3 free magnetic tapes)

Processes	Maximum Needs	Current Needs
P1	10	5
P2	4	2
P3	9	2

So at time t0, the system is in a safe state. The Process P2 can immediately be allocated all its tape drives and then return them. After the return the system will have 5 available tapes, then process P1 can get all its tapes and return them (the system will then have 10 tapes); finally, process P3 can get all its tapes and return them (The system will then have 12 available tapes). The sequence is <P2, P1, P3> satisfies the safety condition.

A system can go from a safe state to an unsafe state. Suppose at time t1, process P3 requests and is allocated one more tape. The system is no longer in a safe state. At this point, only process P2 can be allocated all its tapes. When it returns them the system will then have only 4 available tapes. Since P1 is allocated five tapes but has a maximum of ten so it may request 5 more tapes. If it does so, it will have to wait because they are unavailable. Similarly, process P3 may request its additional 6 tapes and have to wait which then results in a deadlock.

The mistake was granting the request from P3 for one more tape. If we made P3 wait until either of the other processes had finished and released its resources, then we could have avoided the deadlock

Note: In a case, if the system is unable to fulfill the request of all processes then the state of the system is called unsafe.

The main key of the deadlock avoidance method is whenever the request is made for resources then the request must only be approved only in the case if the resulting state is a safe state.

Deadlock Detection and Recovery in OS

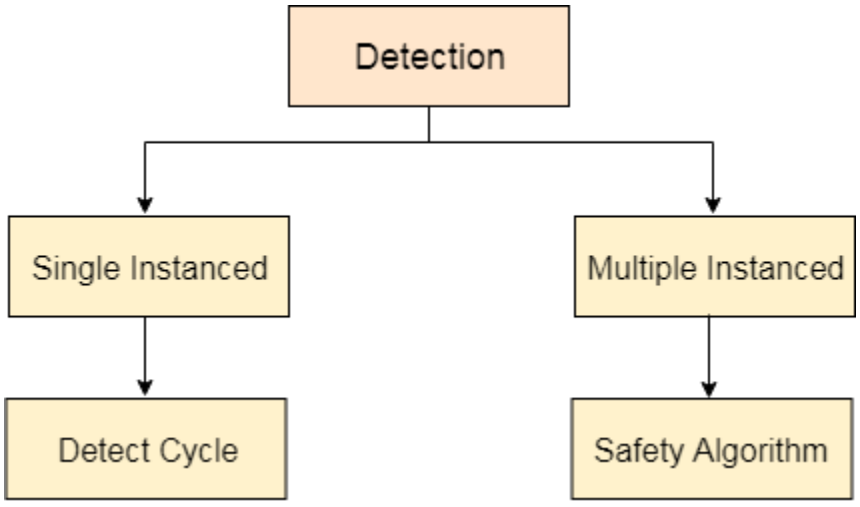
If a system does not employ either a deadlock-prevention or deadlock-avoidance algorithm, then there are chances of occurrence of a deadlock.

In this case, the system may provide two things:

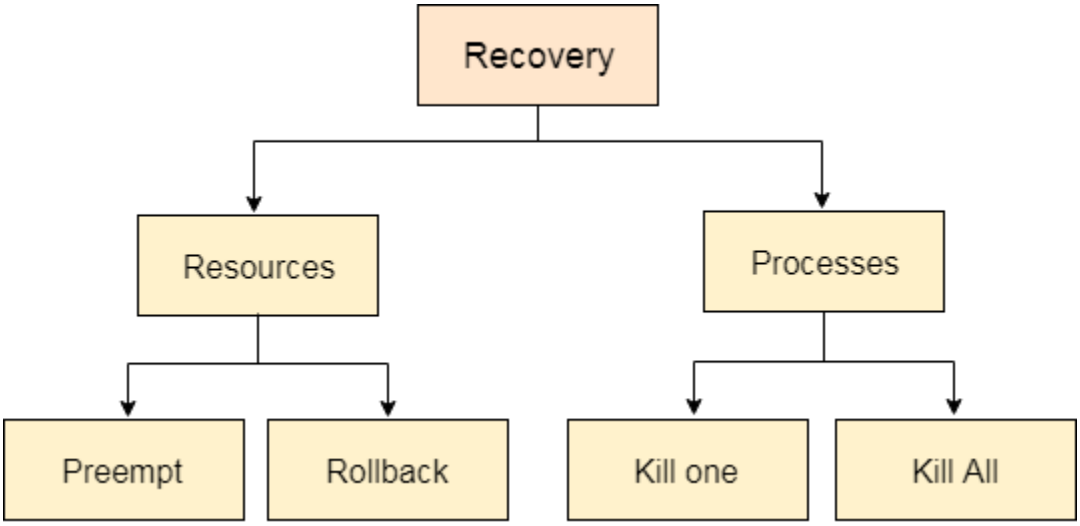
- An algorithm is used to examine the state of the system in order to determine whether a deadlock has occurred.
- An algorithm that is used to recover from the deadlock.

Thus order to get rid of deadlocks the operating system periodically checks the system for any deadlock. After finding the deadlock the operating system will recover from it using recovery techniques.

Now, the main task of the operating system is to detect the deadlocks and this is done with the help of Resource Allocation Graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graphs, detecting a cycle is not just enough. We have to apply the safety algorithm on the system. In order to recover the system from deadlocks, either OS considers resources or processes.



For Resource

Preempt the resource: We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state: System passes through various states to get into the deadlock state. The operating systems can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

For Process

Kill a process: Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process: This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

➢ Computer security

Operating systems employ security and protection measures to prevent a person from illegally using resources in a computer system, or interfering with them in any manner. These measures ensure that data and programs are used only by authorized users and only in a desired manner, and that they are neither modified nor denied to authorized users. *Security* measures deal with threats to resources that come from outside a computer system, while *protection* measures deal with internal threats.

➢ Overview of Security And Protection

A resource could be a hardware resource such as an I/O device, a software resource such as a program or data stored in a file, or a service offered by the OS. Several kinds of interference can arise during operation of a computer system; we call each of them a threat. Some of the threats depend on the nature of specific resources or services and the manner of their use, while others are of a generic nature.

Operating systems use two categories of techniques to counter threats to data and programs:

- ***Security*** measures guard a user's data and programs against interference from persons or programs outside the operating system; we broadly refer to such persons and their programs as nonusers.
- ***Protection*** measures guard a user's data and programs against interference from other users of the system.

Table 15.1 Terminology Used in Security and Protection of Information

Term	Explanation
Authentication	Authentication is verification of a user's identity. Operating systems most often perform authentication *by knowledge*. That is, a person claiming to be some user X is called upon to exhibit some knowledge shared only between the OS and user X, such as a password.
Authorization	Authorization has two aspects: 1. Granting a set of access privileges to a user; for example, some users may be granted read and write privileges for a file, while others are granted read-only privileges, 2. Verifying a user's right to access a resource in a specific manner.

Table 15.1 describes two key methods used by operating systems for implementing security and protection. *Authentication*, which is aimed at security, consists of verifying the

identity of a person. Computer-based authentication rests on either of two kinds of assumptions. One common assumption is that a person is the user he claims to be if he knows something that only the OS and the user are expected to know, e.g., a password. It is called *authentication by knowledge*. The other authentication method relies on things that only the user is assumed to possess. For example, *biometric authentication* is based on some unique and inalterable biological feature such as fingerprints, retina, or iris.

Authorization is the key method of implementing protection. It consists of: (1) granting an *access privilege* for a resource to a user, which is a right to access the resource in the specified manner, and (2) determining whether a user possesses the right to access a resource in a specific manner.

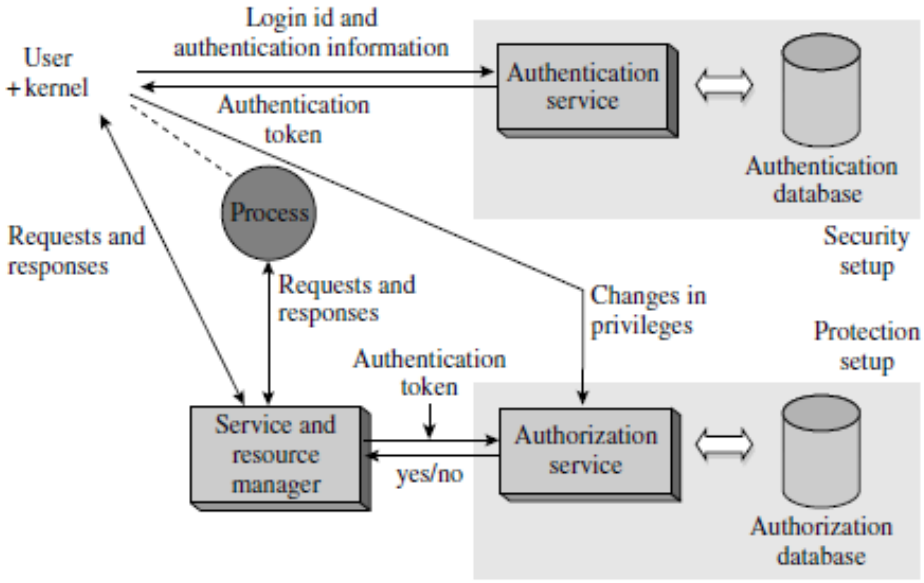


Figure 15.1 Generic security and protection setups in an operating system.

Figure 15.1 shows a generic scheme for implementing security and protection in an operating system. The security setup is shown in the dashed box in the upper part of the figure. It consists of the *authentication service* and the *authentication database*. The authentication database contains a pair of the form (login id, validating information) for every registered user of the operating system, where the validating information is typically an encrypted form of a user's password. To log into the system, a person submits his login id and password to the kernel. The kernel passes this information to the authentication service, which encrypts the password and compares it with the validating information for the user stored in the authentication database. If the check succeeds, the authentication service generates an *authentication token* for the user and passes it back to the kernel. The authentication token is typically the user id assigned to the user. Whenever the user or a

process initiated by the user makes a request to access a resource, the kernel appends the user's authentication token to the request to facilitate making of protection checks.

The protection setup is shown in the dashed box in the lower part of Figure 15.1. It consists of the authorization service and the authorization database. The authorization database contains triples of the form (authentication token, resource id, privileges). When a user wishes to grant access privileges for one of his files to some users, or withdraw some previously granted access privileges for the file, he makes a request to the kernel. As shown in Figure 15.1, the kernel passes on the request to the authorization service along with the authentication token for the user. The authorization service now makes appropriate changes in the authorization database. To access a resource, a user or his process makes a resource request to the service and resource manager. The request contains the id of a resource, the kind of access desired to it, and the authentication token of the user. The service and resource manager passes the request to the authorization service, which determines whether the user possesses the privilege to use the resource in the desired manner and sends a yes/no reply to the service and resource manager. Depending on this reply, the service and resource manager decides whether the user's request should be granted.

Not all operating systems incorporate all the elements shown in Figure 15.1 in their security and protection setups. For example, in most modern operating systems, the authorization information is typically maintained and used by the file system, so the operating system does not maintain the authorization database and does not perform authorization.

The distinction between security and protection provides a neat separation of concerns for the OS. In a conventional operating system, the security concern is limited to ensuring that only registered users can use the system. A security check is performed when a person logs in. It decides whether the person is a user of the OS and determines his user id. Following this check, all threats to information stored in the system are protection concerns; the OS uses the user id of a person to determine whether he can access a specific file in the OS. In a distributed system, however, security concerns are more complex because of the presence of the networking component.