# Lecture 03: Object and Directory Management

This lecture discusses how the operating system manages objects stored in the system. It will proceed under the following captions:

- File Concept
- Review of Storage Devices
- Operations on OS Objects
- Device Directory
- Review of File Access Methods
- Allocation Methods
- Directory Systems
- Innovations and Deviations from Standard Approaches
- Object Protection
- Summary and Concluding Remarks

## 3.1 File Concept

A file may be defined as a collection of related data (records) defined by its creator. An alternate definition of a file is a sequence of bits, bytes, lines or records whose meaning is defined by its creator and user.

Files represent programs (source and object), data, as well as other objects. Files may be free format (e.g. text files), or rigidly formatted (e.g. database files).

The user relates to a logical unit that represents the file; the OS maps this representation to the physical memory and devices.

Properties of a file stored by the OS include name, type, time of creation, creator, record length, size etc.

On some modern systems, the term object is preferred to file. In some systems, all objects are seen as files; in others, the object type is used as a distinguishing feature.

**Example 1:**

For systems such as DOS, OS-1, OS-2, Windows:
All objects in a directory are files. Type is determined by the extension as well as how the object is used.

For Unix, AIX, Pick, Linux:
All objects in a directory are files. The directory is also a file. Type is also determined by the extension as well as how the object is used.

For IBM i:
The library is the holding area of system objects and is itself an object. The object type is used as a distinguishing feature. A library may therefore hold different types of objects.

## 3.2 Revision of Storage Devices

The main storage devices are:
- Magnetic Tape
- Magnetic Drum
- Magnetic Disks
- Optical Disks
- Flash Drive
- Zip Drive
- Video (for inputs)

The student is advised to review how these operate. Note that access of these devices is dictated and controlled by the OS (via privileged instructions).

## 3.3 Operations on OS Objects

The following operations are permissible on OS objects:
- Create
- Modify
- Delete
- Review

In the case of database files, text files and programs, records may be
- Added
- Updated
- Reviewed
- Deleted

All operating systems provide these basic operations.

## 3.4 Device Directory

A device directory is a file directory of all files stored on a given device. This is maintained by the OS. Information which may be included in a device directory are as follows:
- File name
- File type
- Location — a pointer to the device location
- Size — in bytes or blocks
- Current position — a pointer to the current read/write position in the file
- Protection data
- Usage count — No. of processes currently using this file
- Creation date and time, last modification date an time

## 3.4 Device Directory (continued)

**Alternatives for Storing the Device Directory:**

The device directory may be stored in any of the following ways:

- Linear List or Linked List
  - Time consuming to search
  - Easy to maintain

- Sorted List
  - Binary search is fast, but
  - Algorithm is difficult to program
  - List must be kept sorted

- Binary Tree
  - Fast search, but
  - Structure of tree depends on order of insertion
  - Expensive to constantly balance the tree after insertions

- B-Tree
  - Fast search, but
  - Algorithm is even more difficult to program than binary tree
  - Tree must be maintained

- Hash Table
  - Very fast search
  - Insertion & deletion fairly straightforward but provision for collisions must be made
  - Fixed size of hash table and the dependence of the hash function on the size of the hash table are the two major difficulties. The hash table must therefore be sufficiently large.
  - Another problem is that of collision resolution. Linear probing and rehashing are both unacceptable for disks over 90% full. Synonym chaining (or open addressing with buckets) is acceptable, but again memory space is not infinite.

## 3.5 Review of File Access Methods

The principal file access methods are:
- Sequential access
- Direct/Random access
- Indexed Sequential Access
- Multi Key access

Please review these access methods. Note that implementation of a given access method is a function of the OS.

## 3.6 Allocation Methods

How does the OS allocate space to store its objects? A number of approaches exist:

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation
- Composite System
- Hash Coded Files

In order to proceed we must spend some time addressing the issue of free space management.

### 3.6.1 Free Space Management

The operating system maintains a *free space list*. This could be implemented via a bit map, where one bit representing each block (1 denoting an occupied block, 0 denoting an empty block).

To create a file (an object), the free space list is searched to identify a large enough area for that object. The object is placed there, and the relevant blocks taken from the free space list. To delete an object, the space taken up is returned to the free space list.

To improve the system, each item in the free space list may be recorded as a table in the following way:

| **Block Address** | **No. of Free Consecutive Blocks Following** |
|---|---|
| 1111 | 10 |
| ... | ... |
| FFFF | 20 |

This would aid in slotting the objects based on their respective sizes.

### 3.6.2 Contiguous Allocation

In *contiguous allocation*, each object (file) must occupy a set of contiguous blocks (addresses) on the disk. An object is defined by its block address, **b**, and the number of blocks, **n**, it occupies. Thus if an object is defined by **b**, **n**, then it starts at block **b** and occupies a space up to block $\mathbf{b}_{n-1}$.

Access of a file stored under contiguous allocation may be sequential or direct. Numbering of blocks may be per disc or along cylinders in which case, the heads move less.

Finding contiguous free space:
- Use the bit map: for an n-block file, find n consecutive 0's in the bit map.
- Use the free space table: find a table item where the number of contiguous blocks is n.

How is free space allocated? Three strategies are common:
- First fit: allocate the first hole that is big enough.
- Best fit: allocate the smallest hole the object can fit in.
- Worst fit: allocate the largest hole that the object can fit in.

Simulations show that first fit and best fit are preferred to worst fit in terms of time and storage utilization.

Hashing is sometimes used with contiguous allocation: A hash function generates the address of the file which is then stored contiguously.

Contiguous allocation algorithms suffer from "external fragmentation":
- As objects are created and deleted, the free space is broken up into little pieces.
- External fragmentation occurs when enough disk space exists to meet the request, but the space is not contiguous.

Compaction is the method used to address the problem of external fragmentation.
- The objects are temporarily copied to another disk (area);
- The objects are removed from the original disk area, creating a large hole;
- The objects are then copied back to the original disk (area).

Contiguous allocation and compaction were popular with microcomputer systems running DOS.
Example: the DOS/Windows command DEFRAG performs the compaction function.

Advantages of contiguous allocation include:
- Minimum movement of disk heads therefore high processing speed.
- Easy direct access possible.
- Directory entries for an object is small — simply fist block and length.
- As object size increases, the OS automatically finds a large enough space for it to fit.

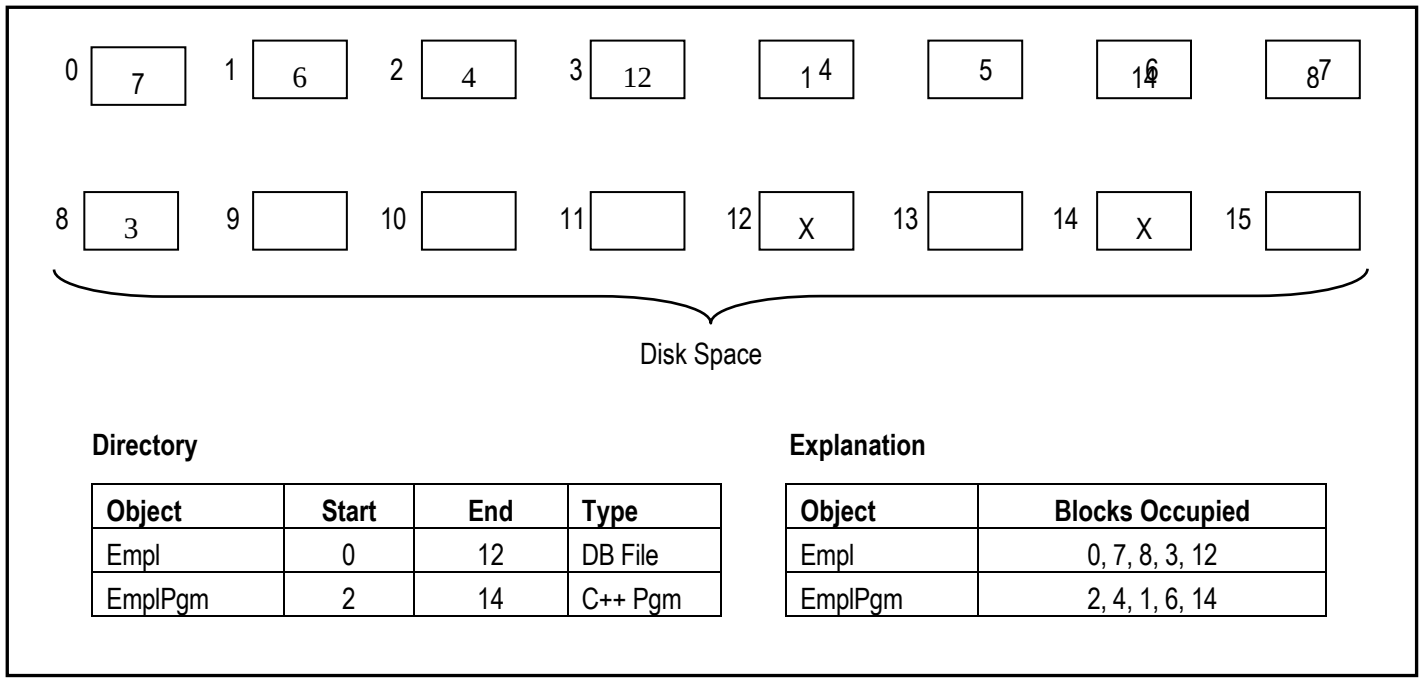
### 3.6.2 Contiguous Allocation (continued)

Disadvantages of contiguous allocation include:
- Object size must be known before it is written. This is quite clumsy.
- Due to this size limitation, space may be wasted, which contributes to external fragmentation.
- Object (file/program) modification will affect object size, thus resulting in either space underutilization or reallocation of the object. This also contributes to external fragmentation.

### 3.6.3 Linked Allocation

In *linked allocation*, the OS takes the first free space available and starts writing the (first) object there. A pointer points to the next free space (sector or block) and so on. Each object is therefore a linked list of disk space (sectors / blocks) Figure 3.1 illustrates.

**Figure 3.1: Illustrating Linked Allocation**




**Directory**

| Object | Start | End | Type |
|---|---|---|---|
| Empl | 0 | 12 | DB File |
| EmplPgm | 2 | 14 | C++ Pgm |

**Explanation**

| Object | Blocks Occupied |
|---|---|
| Empl | 0, 7, 8, 3, 12 |
| EmplPgm | 2, 4, 1, 6, 14 |

The essential directory entries for an object would be:
- Object name
- Starting address
- Ending address

The pointers used to link data blocks are not available to the user; only the OS.

**Example 2:** If each sector is 512 bytes and the pointer is 2 bytes long, the user sees 510 bytes per sector.

### 3.6.3 Linked Allocation (continued)

To create an object, a new entry is added to the device directory. On deletion the device directory entry is removed. To read an object, the linked list is followed (sequential access).

Advantages of linked allocation include:
- Elimination of external fragmentation.
- Easy file creation
- Object size is not needed in advance
- Object size can grow as necessary (middle or end)
- Returning free space is easy (simply remove the directory entry and update the free space list)

Disadvantages of linked allocation include:
- Suitable for sequential access only
- Slow access since the disk heads have to move quite often
- Space required by pointers in each sector or block
- If pointer or a sector or block is damaged, access is impossible

### 3.6.4 Indexed Allocation

*Indexed allocation* utilizes an index block consisting of pointers to the data blocks. Each object (file) has its own index block which is an array of disk block addresses. The $i^{th}$ entry in the index block contains addresses of the $i^{th}$ block of the object (file).

The device directory contains a pointer to the index block of each object. At creation, pointers in the index block for a given object are initially null (until the object size warrants them being non-null).

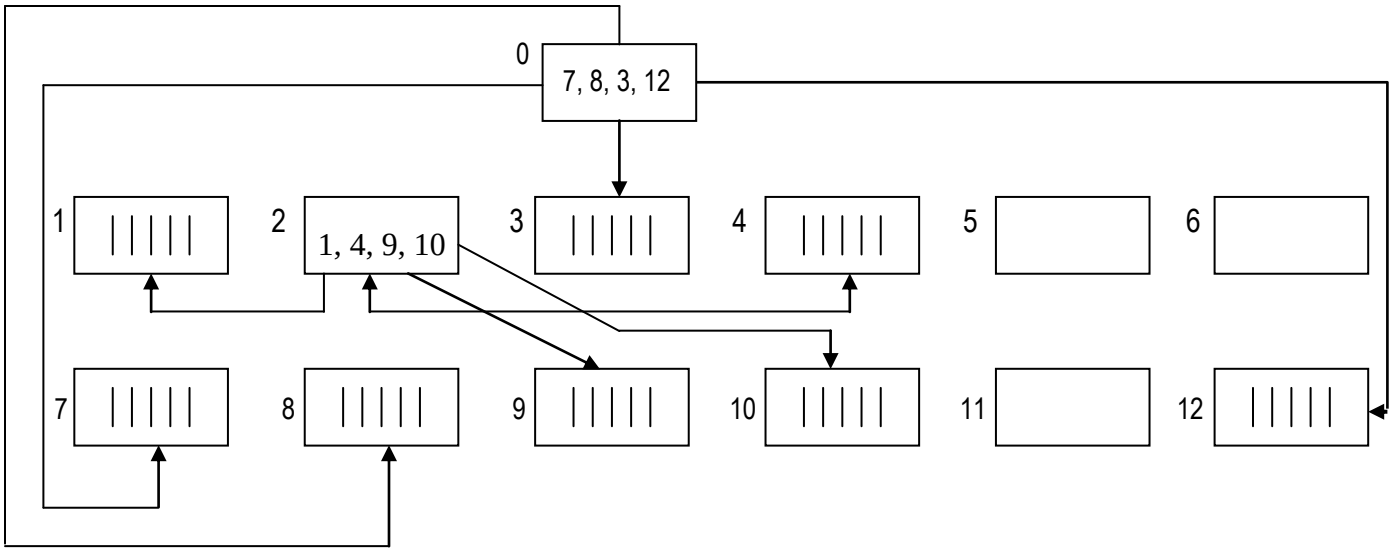
When the $i^{th}$ block is written (for the first time), a block is removed from the free space list and its address put in the $i^{th}$ index block entry. When a block is deleted, it is simply removed from the index block and returned to the free space list.

Direct access is supported by simply using the index to access the specific data blocks required. Through the index, the file can also be accessed sequentially.

The OS-2 system of the 1990s implemented a variation of indexed allocation, combined with contiguous allocation. Called the *High Performance Filing System* (HPFS), this system sought to minimize fragmentation by allowing newly created files to be scattered in bands across the disk surface, in a manner that avoids interleaving. Another strategy was to allocate 4KB of free space to each file that needs to be extended; if the space is not used, the remainder is returned to the free space. Indexes were implemented via B trees, B+ trees, as well as bitmaps.

### 3.6.4 Indexed Allocation (continued)

**Figure 3.2: Illustrating Indexed Allocation**




**Directory**

| Object | Index Block |
|---|---|
| Empl | 0 |
| Empl-pgm | 2 |

**Explanation**

| Object | Index Block | Data Blocks |
|---|---|---|
| Empl | 0 | 7,8,3,12 |
| Empl-pgm | 2 | 1, 4, 9, 10 |

Advantages of indexed allocation include:
- Elimination of external fragmentation
- Facilitation of direct access as well as sequential access

Disadvantages of indexed allocation include:
- Space wasting is possible within an index block (if the block is larger than is required for that object).
- One index block might not be sufficient for large objects.

These drawbacks may be offset (but not without increasing the complexity of the access algorithm) by any of the following strategies:

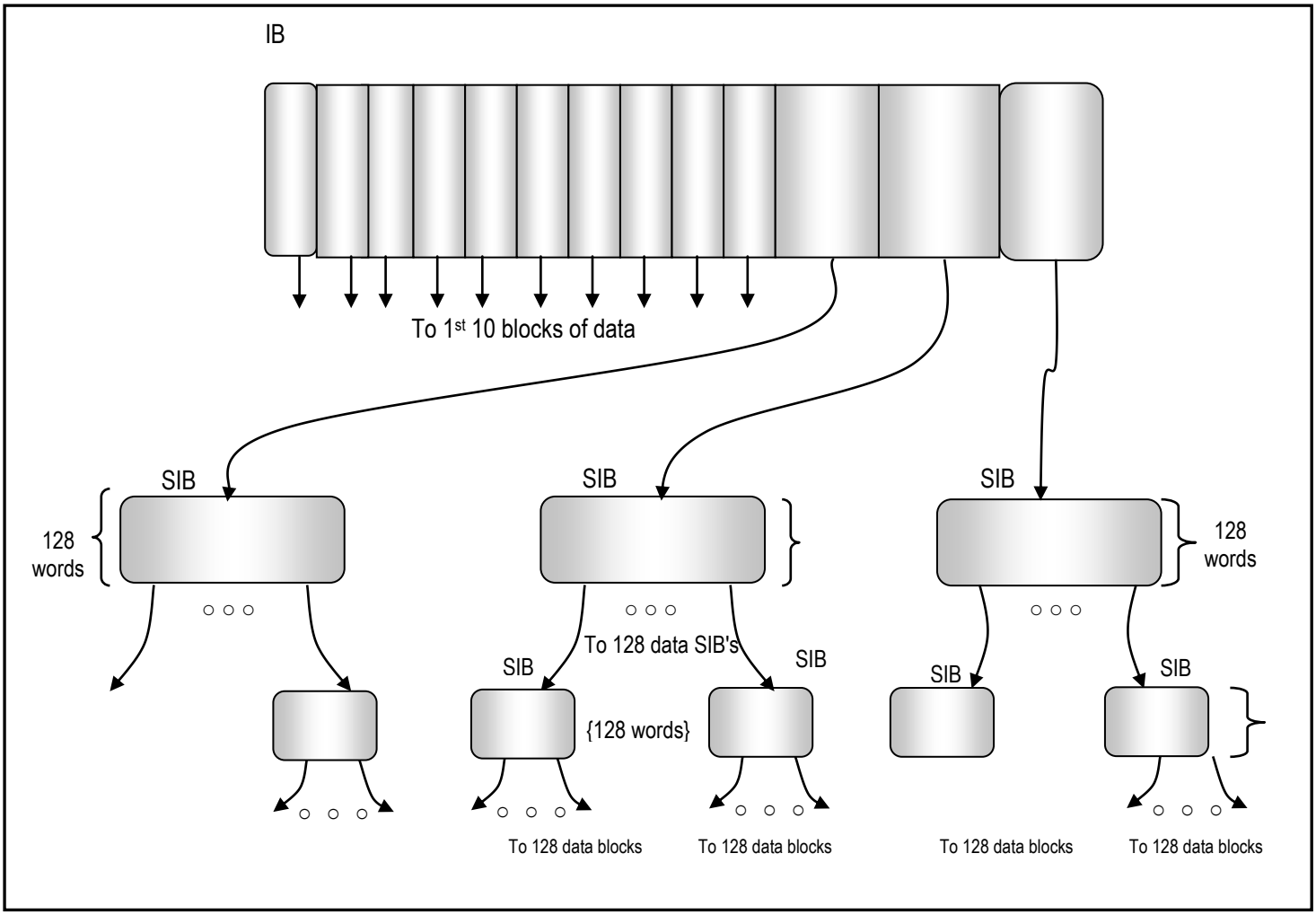
a. Make index blocks of moderate size. If necessary, an index block may point to another index block and so on....
b. Use dynamic arrays for index blocks.

### 3.6.5 Composite Systems

The composite system employs aspects from linked allocation and indexed allocation. Having first been implemented on the UNIX operating system, it is the most widely used system for contemporary operating systems. Figure 3.3 illustrates an implementation of this approach.

At level 1 (highest level), a fixed number of index block (IB) entries point to data blocks. The other IB entries point to level 2 IB's. Level 2 IB's point to data blocks (or level 3 IB's if necessary)… and so on.

**Figure 3.3: Illustrating Composite System**




The significant advantages of the composite system are:

- Maximization of the advantages of the earlier approaches
- Minimization of the disadvantages of the earlier approaches
- Facilitation of large files as well as small files

The main drawback is that the programming requirements are complex.

### 3.6.6 Hash-Coded File Allocation

For a hash-coded filing system, there is no device directory. A given mapping function takes the file name and calculates some address which corresponds to the file name. Hashing may be used in conjunction with indexed allocation to determine the first IB of the file.
It may be used in conjunction with linked allocation and contiguous allocation.

Advantages of hash-coded filing systems are:
- Facilitation of direct access and sequential access
- Fast response

Disadvantage of hash-coded filing systems are:
- The whole disk space may have to be searched to locate a file or determine its non-existence. To avoid this problem, the hash addresses can be pre-calculated (and any collision resolved) and stored in a hast-table.
- Collision resolution is potentially problematic. Linear probing and rehashing are both unacceptable for disks over 90% full. Synonym chaining (or open addressing with buckets) is acceptable, but again disk space is not infinite.

Hashing has been successfully used on PICK OS (in concert with contiguous allocation).

## 3.7 Directory Systems

As the amount of storage and number of users increase, it becomes increasingly difficult to track objects on the computer. The solution to this problem is the imposition of a directory structure on the file (object) system. A directory system provides a mechanism for organizing system objects (and files).

Many systems have two separate directory structures. The device (physical) directory (section 3.4) and the file (logical) directory. We will focus on the latter here.

Each object on the system has two names: the user name and the OS name. The OS provides the mapping between user names and OS names; between the logical directory system and the physical location of the object.

Information stored in the directory includes:
- Object name
- Object type
- Object owner
- Creation date
- Modification date (last modification)
- Access rights
- Object size
- Accounting information

## 3.7 Directory Systems (continued)

Operations which can be performed on a directory include:
- Creation of directory
- Deletion of directory
- Creation of objects to be contained in the directory
- Modification of objects in the directory
- Deletion of objects from the directory
- Listing contents of the directory
- Directory backup
- Moving objects from one directory to another

On some systems, the term "library" is preferred to "directory". The System i (formerly OS-400) is one prominent example. Contemporary Windows platforms prefer the term “folder”. Also, on some systems there is a limit to the number of (database) files that can be simultaneously opened.

Some systems (e.g. System i) allow for object types in the directory; others treat all objects as files and their use depends on the user's interpretation (e.g. Unix, PICK, Windows).

Advantages of file/object types:
- The OS treats each object differently based on type.
- By use of name, it is clear what the purpose of the object is.

Disadvantages of file/object types:
- Everything is forced into a given pattern.
- Certain commands apply to certain object types only, thus learning may be more challenging.

The directory structure may be any of the following:
- Single-Level Directory
- Two-Level Directory
- Multi-Level (Tree Structured) Directory
- Acyclic Graph Directory

### 3.7.1 Single-Level Directory

In a single-level directory system, all objects are in the same directory. It is very easy to support and understand.

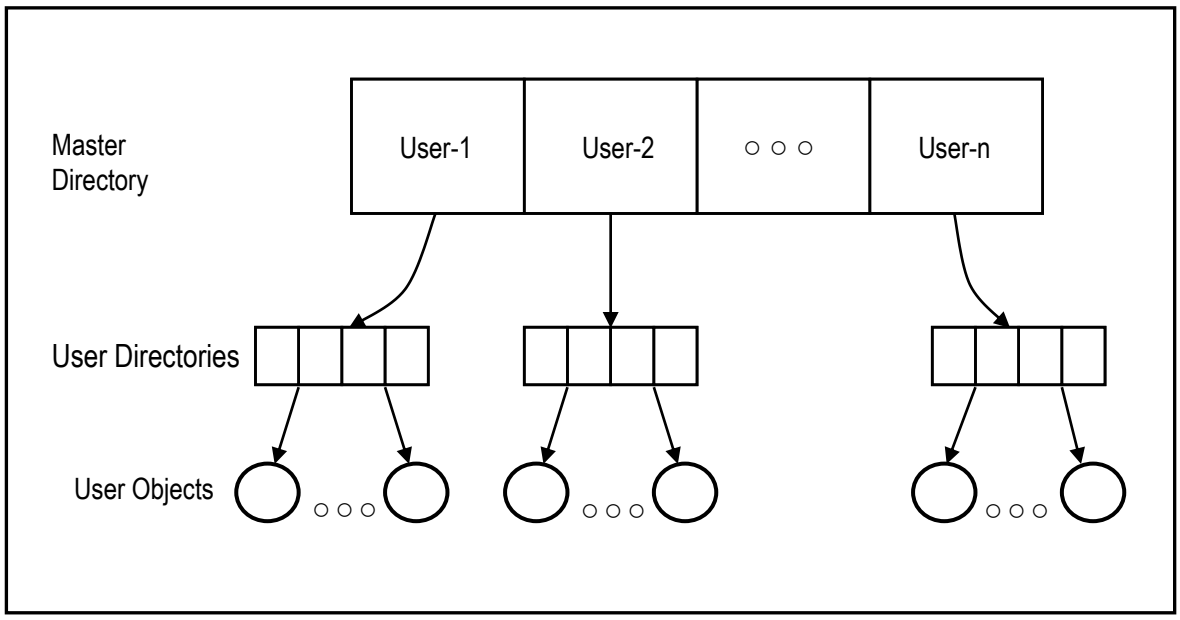
The main drawbacks of the approach are:
- Object names must be unique. This could be a challenge when the number of objects increases and/or the number of users increases.
- Object categorization becomes difficult.

The approach was used on early single user systems.

### 3.7.2 Two-Level Directory

In a two-level directory system, each user has his/her own directory as illustrated in figure 3.4. Additionally, there is a system directory.

**Figure 3.4: Illustrating Two-Level Directory Structure**




Advantages of the approach:
- Different users may have same object-names
- Improved directory search
- Improved object security

Disadvantages of the approach:
- Requires more disk space (not a real problem nowadays)
- Difficult to share objects. Additional instructions and strategies must be introduced
- Access to system-wide objects is a problem

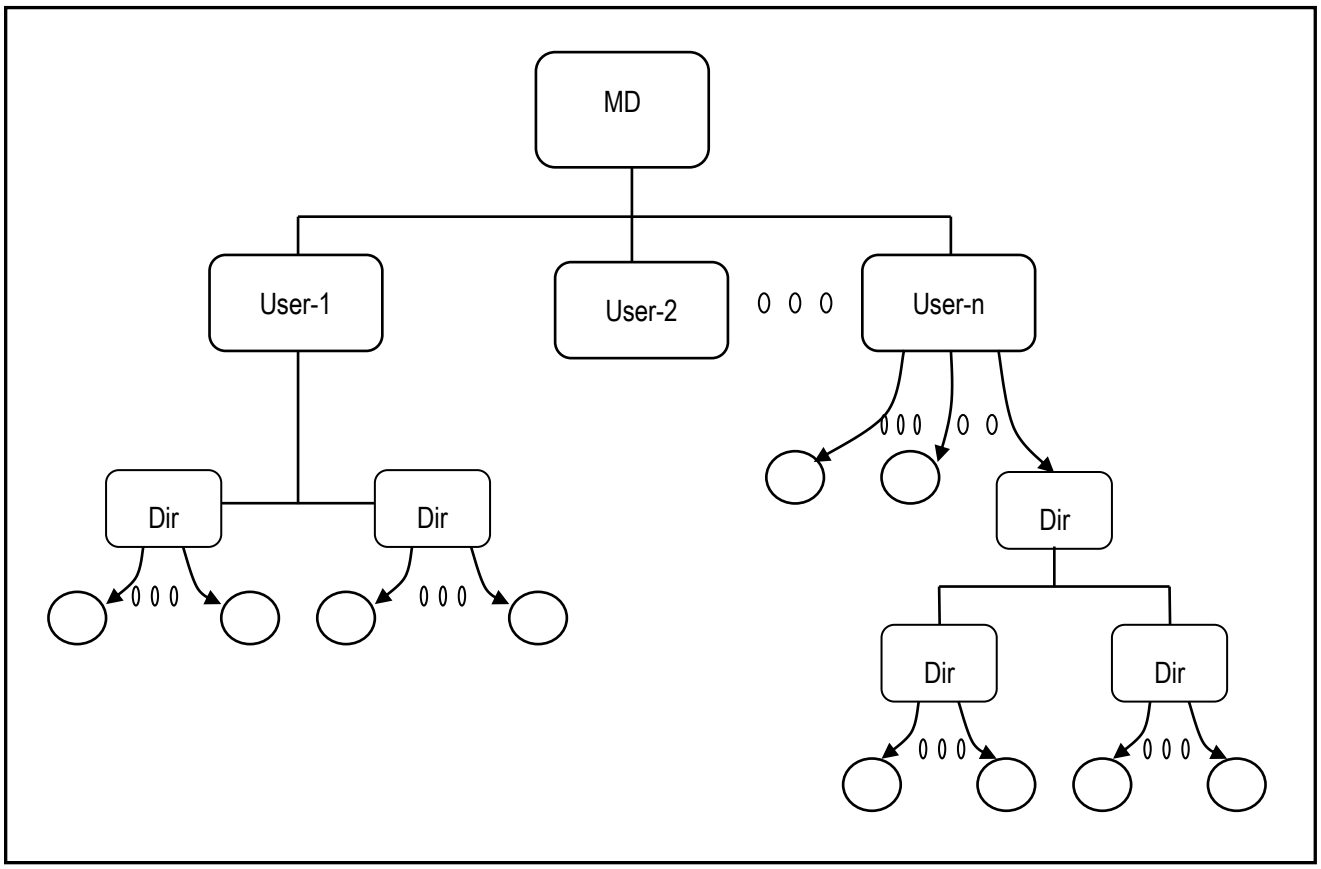
Possible solutions to these drawbacks:
- With technological development, memory is not a real problem.
- Use of pathnames and access rights.
- System-wide objects are (usually) put in special purpose directories which are accessible to all users.

The approach has been successfully implemented on the PICK OS, where directories were called "accounts".

### 3.7.3 Multi-Level (Tree Structured) Directory

The multi-level directory system is a generalization of the two-level directory, users can create subdirectories indefinitely. It has been successfully implemented on operating systems such as Unix, DOS, Windows, OS/2, and Linux.

**Figure 3.5: Illustrating Multi-Level Directory**




**Note:**

- The system must distinguish between directories and other objects (files), hence the importance of object types.
- Access paths and access permission codes allow for cross-referencing.

Deletion of a directory must follow a specific method. Two approaches are common:

- Delete all objects and subdirectories of that directory, then delete the directory.
- Disallow deletion of a non-empty directory.

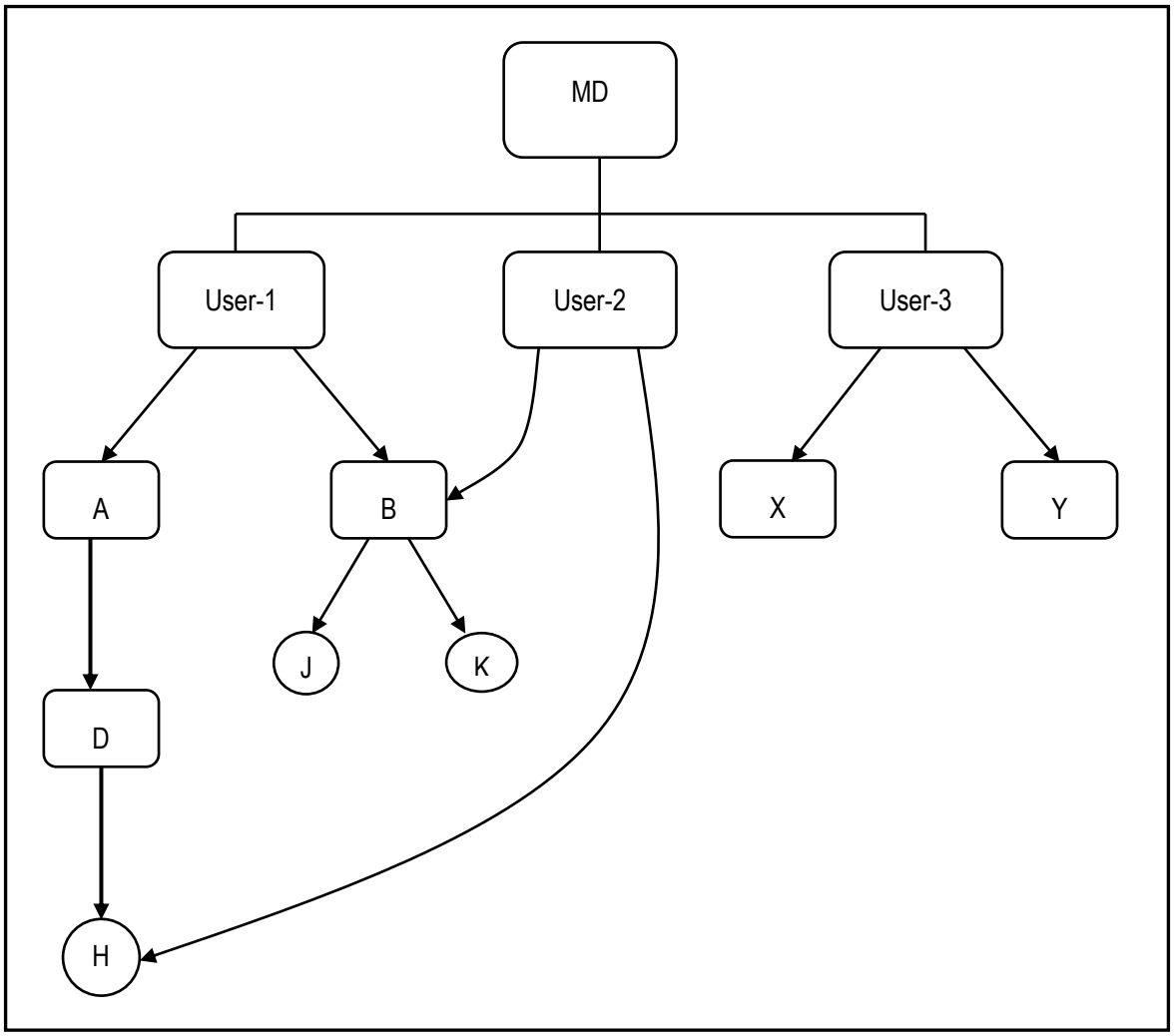
### 3.7.4 Acyclic Graph Directory

The main drawback with the two-level and multi-level approaches is the clumsy way of facilitating group work as well as integrated system development (to amplify the point, simply consider two programmers writing programs that use a common set of files).

An acyclic graph allows for the sharing of objects across directories and subdirectories themselves. The case of several people working on a project is solved by simply putting the shared objects in one directory.

Figure 3.6 illustrates the approach. Directories and objects may be linked as long as a closed loop is not described. Also, different pathnames may point to the same object.

**Figure 3.6: Illustrating Acyclic Graph**



### 3.7.4 Acyclic Graph Directory (Continued)

**Example 3:**

In figure 3.6, object H may be described by any of the following paths:
- MD\USR-1\A\D\H
- MD\USR-2\H

Still referring to figure 3.6, deletion of objects may be treated in one of the following ways:
- If usr-2 no longer needs to use H, simply remove the link to H. The object H may only be deleted if there are no (more than one) pointers to it.
- If H is to be deleted, remove all pointers to it.

The latter of the two approaches is extremely dangerous and is therefore not recommended.

Drawback with Acyclic Graph: The OS must keep track of links to ensure that loops are not formed. This increases the level of difficulty in writing the OS.

Examples of acyclic graphs: MS Windows via its association and object linking facilities, as well as network shares; Unix via its **link** command.

## 3.8 Innovations and Deviations from Standard Approaches

While some systems adhere to the standard (traditional) approaches as presented, others tend to have innovations of the basic approaches.

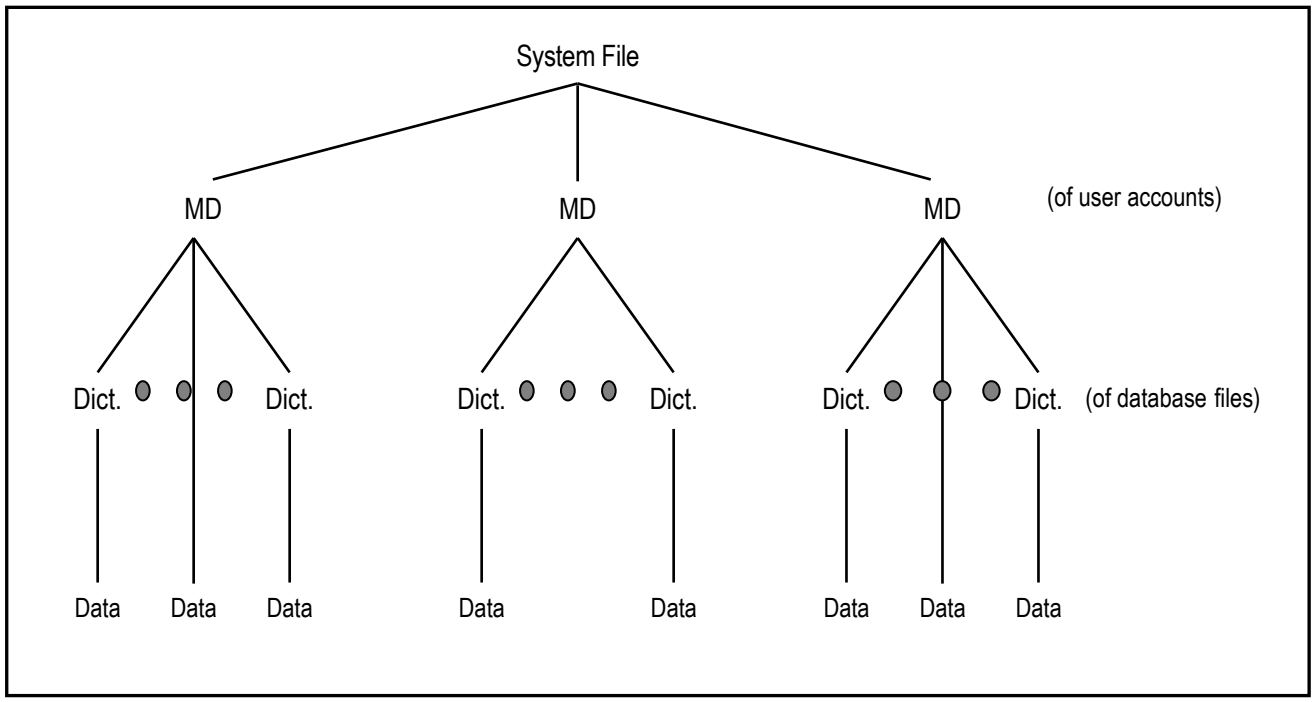
DOS, Unix and Windows follow the tree structured approach (by extension, OS-1, OS-2 and AIX are also tree structured). MS-Windows, Unix, and Linux facilitate acyclic linkages in a tree structured directory system.

### 3.8.1 PICK Deviation

The PICK system implemented an innovation of the two-level directory system as follows:
- A system file stores information on all users of the system.
- Each user has an account.
- Each account has a master directory (MD) of all files in that account.
- Files are seen by the system as a collection of bytes (and bits). As such, they may be variable length and support variable length records. File type is not known until the file is being used.
- A database file usually has a dictionary portion which describes the data stored.
- Users sign on to their accounts. The system facilitates referencing (via aliasing, albeit in a very cumbersome and technical manner).
- File allocation is hashing merged with contiguous allocation.

### 3.8.1 PICK Deviation (continued)

**Figure 3.7: Illustrating PICK OS File/Directory Structure**




### 3.8.2 OS-400 Deviation

The IBM i (formerly OS-400) implements an innovation of the two-level directory system, combined with the concept of object-oriented design (OOD) as follows:

- All data is stored as objects of various types.
- The library is the holding area for all system and user-created objects.
- The OS is shipped with a number of important libraries. Some of them contain objects used by the OS, and are called system objects. Others contain objects that users will find useful.
- Each object has an object type; the commonly used types are listed in figure 3.8.
- The OS-400 keeps track of all objects, transparent to the user.
- A user profile (USRPRF) is a special kind of object which identifies a user. All user profiles are stored in a system library, QSYSUSR.
- Users may use objects (in any library) as long as they have the appropriate authority to those objects.
- No object's usage is confined to the library that it resides in; it may be used from anywhere; the library is simply a holding area. Typically, an application indirectly references an object through the operating system. This can be done in several ways, the simplest being to include the library in the user's library list (an attribute that is set when the user profile is created or modified).
- Objects are independent entities, linked only by commands, programs, or the operating system itself. The OS was developed out of the expansion of the concepts of relational systems and object-oriented design.
- An object's library may be explicitly specified, but usually, the library list of the user is modified, in which case the OS searches the library list for the object (library is implicitly specified).
- A library list is initialized in the user profile or on a job description, and may be modified by the commands RPLLIBL, ADDLIBLE, RMVLIBLE.

**Figure 3.8: Some common object types in IBM i**

| | | | |
|---|---|---|---|
| USRPRF: | User Profiles | LIB: | Library |
| PRTF: | Print File | DEVD: | Device Description |
| JOBD: | Job Description | CLS: | User Classification |
| PF: | Physical File | LF: | Logical File |
| CLP: | Control Language Program | OUTQ: | Output Queue |
| RPG: | RPG-400 Program | CMD: | Command |
| .. | | | |
| etc. | | | |

## 3.9 Object Protection

Even with security measures in place (we will discuss this in more detail in lecture 8), objects need to be protected from unintended use.

Each object has certain access rights; namely:
- Read
- Execute (programs only)
- Write
- Review/Display
- Delete

Additionally, for database files, text files and programs, **Read**, **Write**, **Review/Display**, **and Delete** are operations that apply to records in the file.

For each object, the creator has all rights. Additionally, most systems give exclusive rights to all objects, to a *Superior User*. The superior user may have different names on different operating systems
- On PICK the superior user was called the System Manager
- On IBM i, it is called the Security Officer
- On Unix and Linux, it is the Super User
- On Novell, it is the System Supervisor
- On Windows, it is the System Administrator

Other users of the system are given access rights as specified by the object owner or the superior user.

## 3.10 Summary and Concluding Remarks

Here is a summary of what has been covered in this lecture**:**

- The operating system maintains a device directory to keep track of all objects stored on each storage medium.
- The operating system maintains a free space list to keep track of available space on the storage medium.
- The OS uses an object allocation strategy to secure space for objects stored. Among the allocation strategies discussed are contiguous allocation, linked allocation, indexed allocation, composite allocation, and hash-coded files. Contiguous allocation tends to produce a high level of fragmentation. Composite allocation eliminates fragmentation to data within an index block.
- Among the directory systems discussed are single-level directory, two-level directory, multi-level directory, and acyclic graph.
- The IBM i system employs a variation of the two-level directory system that is fully object-oriented and very efficient.

Each OS has a file system that addresses these and other related issues. Two other very important aspects of the file system are I/O management and security. These will be addressed later in the course. The next lecture addresses CPU scheduling.

## 3.11 Recommended Readings

[Bacon & Harris 2003] Bacon, Jean S. & Tim Harris. 2003. *Operating Systems: Concurrent and Distributed Software Design*. Boston, MA: Addison-Wesley. See chapter 6.

[Nutt 2004] Nutt, Gary. 2004. *Operating Systems* 3ed. Boston, MA: Addison-Wesley. See chapter 13.

[Silberschatz 2012] Silberschatz, Abraham, Peter B. Galvin, & Greg Gagne. 2012. *Operating Systems Concepts*, 9th Ed. Update. New York: John Wiley & Sons. See chapter 11–12.

[Stallings 2005] Stallings, William. 2005. *Operating Systems* 5th ed. Upper Saddle River, New Jersey: Prentice Hall. See chapters 11–12.

[Tanenbaum 2006] Tanenbaum, Andrew S., & Albert S. Woodhull. 2006. *Operating Systems: Design and Implementation* 3ed. Upper Saddle River, NJ: Prentice Hall. See chapter 5.

[Tanenbaum 2008] Tanenbaum, Andrew S. 2008. *Modern Operating Systems* 3ed. Upper Saddle River, NJ: Prentice Hall. See chapter 4.