OS Module - V

Storage Management

Part - I

File Systems

Chapter Outcomes

At the end of this chapter, you will be able to;

- Explain file systems.
- To describe the interfaces to file systems.
- To discuss file-system design tradeoffs, including access methods and directory structures.
- To discuss block allocation and free-space management.

Ref: Modern Operating System 4th edition by Andrew S Tanenbaum and Operating System Concept 9th edition by Galvin

File Concept

Essential requirements for long-term information storage:

- ✓ It must be possible to store a very large amount of information.
- ✓ The information must survive the termination of the process using it.
- ✓ Multiple processes must be able to access the information concurrently.

File Concept

- Think of a disk as a linear sequence of fixed-size blocks and supporting reading and writing of blocks.
- Questions that quickly arise:

File Concept

- Think of a disk as a linear sequence of fixed-size blocks and supporting reading and writing of blocks.
- Questions that quickly arise:
  - ✓ How do you find information?

File Concept

- Think of a disk as a linear sequence of fixed-size blocks and supporting reading and writing of blocks.
- Questions that quickly arise:
  - ✓ How do you find information?
  - ✓ How do you keep one user from reading another's data?

File Concept

- Think of a disk as a linear sequence of fixed-size blocks and supporting reading and writing of blocks.
- Questions that quickly arise:
  - ✓ How do you find information?
  - ✓ How do you keep one user from reading another's data?
  - ✓ How do you know which blocks are free?

File Concept

- ✓ We can solve this problem with a new abstraction: ***the file***.

- ✓ Together, the abstractions of processes (and threads), address spaces, and files are the most important concepts relating to operating systems.

File

- **Files** are logical units of information created by processes.
- A disk will usually contain thousands or even millions of them, each one independent of the others.
- Processes can read existing files and create new ones if need be.

File

- Information stored in files must be **persistent**, that is, not be affected by process creation and termination.

- A file should disappear only when its owner explicitly removes it.

- Although operations for reading and writing files are the most common ones.

File

- Information stored in files must be **persistent**, that is, not be affected by process creation and termination.
- A file shou
- expli
-

➢ Each file is a kind of address space, except that they are used to model the disk instead of modeling the RAM.

File System

- Files are managed by the operating system.
- How they are structured, named, accessed, used, protected, implemented, and managed is important component in OS design.
- As a whole, that part of the OS dealing with files is known as the **file system.**

File System

- From the user's standpoint, the most important aspect of a file system is;
  - ✓ How it appears?
  - ✓ What constitutes a file?
  - ✓ How files are named and protected?
  - ✓ What operations are allowed on files?
  - ✓ and so on.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**: The symbolic file name is the only information kept in human readable form.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**:
- ✓ **Identifier**: This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**:
- ✓ **Identifier**:
- ✓ **Type**: This information is needed for systems that support different types of files.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**:
- ✓ **Identifier**:
- ✓ **Type**:
- ✓ **Location**: This information is a pointer to a device and to the location of the file on that device.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**:
- ✓ **Identifier**:
- ✓ **Type**:
- ✓ **Location**:
- ✓ **Size**: The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**:
- ✓ **Identifier**:
- ✓ **Type**:
- ✓ **Location**:
- ✓ **Size**:
- ✓ **Protection**: Access-control information determines who can do reading, writing, executing, and so on.

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- ✓ **Name**:
- ✓ **Identifier**:
- ✓ **Type**:
- ✓ **Location**:
- ✓ **Size**:
- ✓ **Protection**:

Apert from these the table next slide shows some of the other attributes may also exist.

File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Table: Some possible file attributes.

File Naming

- ✓ When a process creates a file, it gives the file a name; and the file can be accessed by the name.

- ✓ The exact rules for file naming vary from system to system, but all current operating systems allow strings of letters as legal file names.

File Naming

- Many operating systems support two-part file names, with the two parts separated by a period.
  - ✓ File extension: indicating characteristics of file
  - ✓ In Unix,
    - file extension is just convention
  - ✓ In windows,
    - file extensions specify which program “owns” that extension; when double clicking, program assigned to it is launched

File Naming

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Table: Some typical file extensions.

File Structure

- Files can be structured in any of several ways.
- Three common possibilities are depicted in Fig.

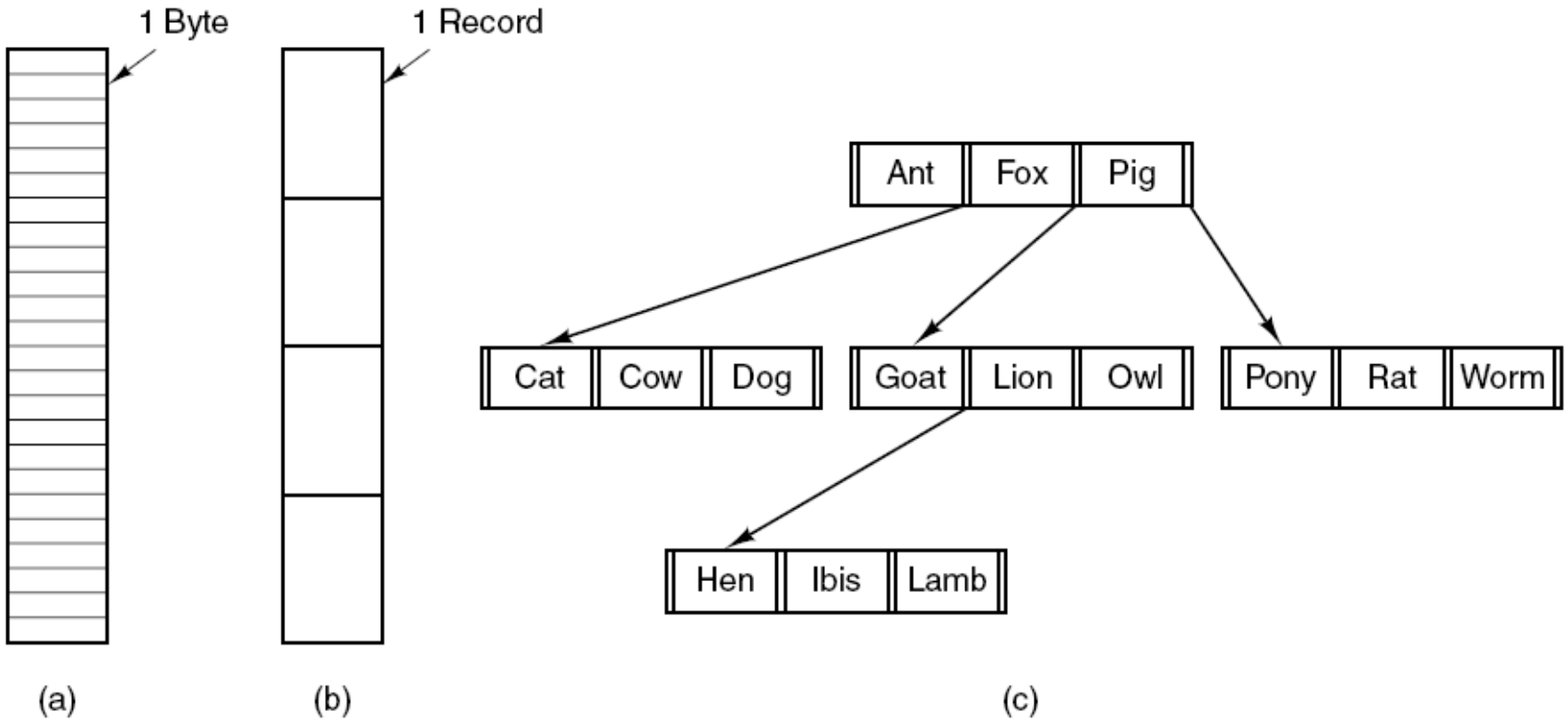


Fig.: Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

File Structure

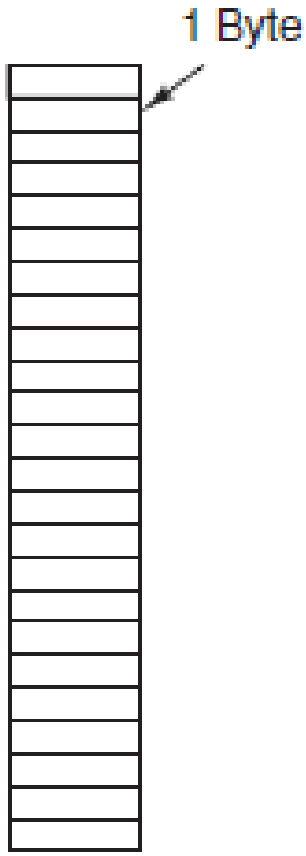


Fig.: Byte sequence

- The file structure in Fig.
  - Is an unstructured sequence of bytes.
  - In effect, the OS does not know or care what is in the file.
  - All it sees are bytes.
  - Any meaning must be imposed by user-level programs.
  - Provides the maximum amount of flexibility.
  - Both UNIX and Windows use this approach.

File Structure

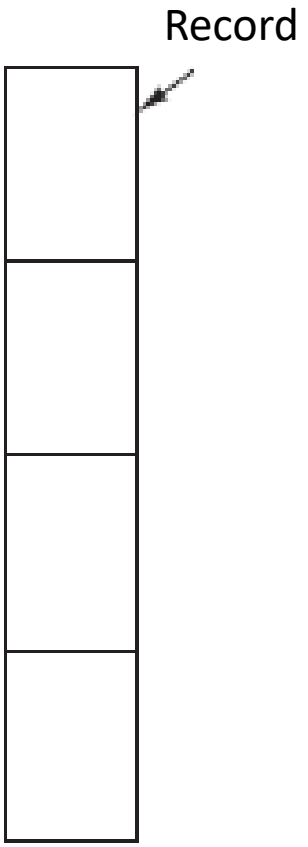


Fig.: Record sequence

- The file structure in Fig.
  - In this model, a file is a sequence of fixed-length records, each with some internal structure.
  - Central to the idea of a file being a sequence of records is the idea that the read operation return some record and the write operation overwrites or appends one record.
  - No current general-purpose system uses this model as its primary file system any more.

File Structure

- The third kind of file structure is shown in Fig.
  - In this organization, a file consists of a tree of records, not necessarily all the same length, each containing a **key** field in a fixed position in the record.
  - The tree is sorted on the key field, to allow rapid searching for a particular key.
  - Used on some large mainframe computers for commercial data processing.

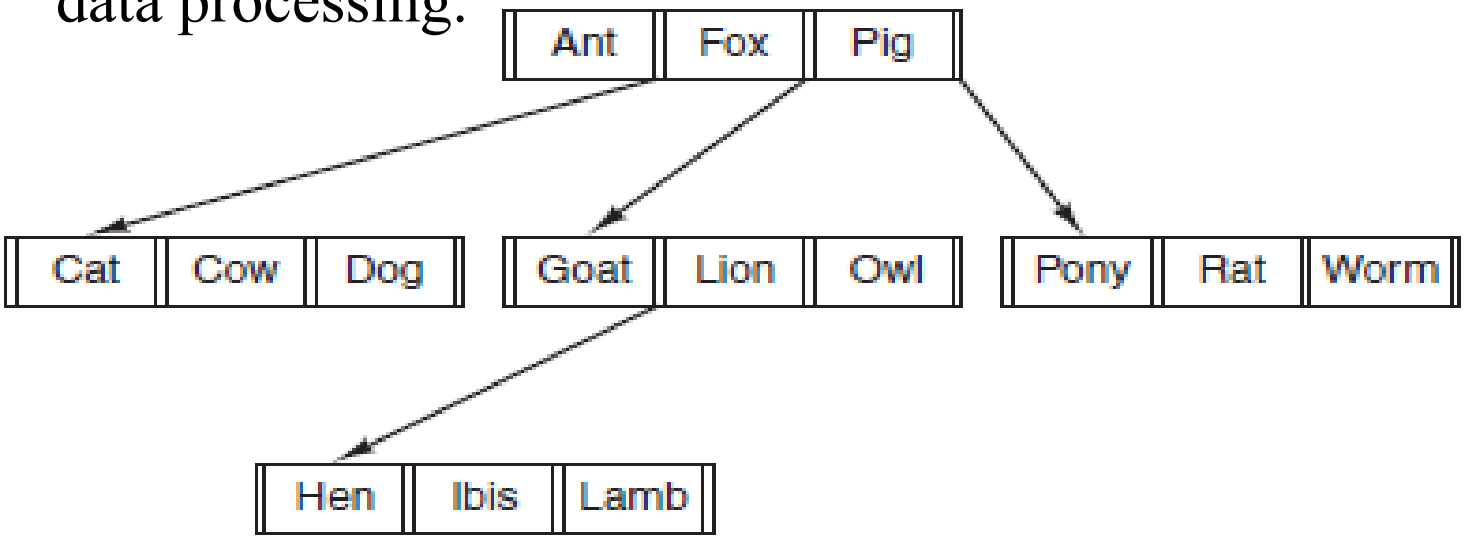


Fig.: Tree

File Types

- Many operating systems support several types of files.
- UNIX and Windows, for example, have regular files and directories.
- UNIX also has character and block special files.

File Types

- **Regular Files**: are the ones that contain user information and are generally either ASCII files or binary files.
  - ASCII files or binary files
  - ASCII consists of lines of text; can be displayed and printed
  - Binary, have some internal structure known to programs that use them

File Types

- **Regular Files**:
- **Directory**: are system files
  - maintains the structure of the file system.

File Types

- **Regular Files**:
- **Directory**:
- **Character special files:** (a character device file)
  - Related to I/O and model serial I/O devices

File Types

- **Regular Files**:
- **Directory**:
- **Character special files:**
- **Block special files:** (a block device file)
  - Mainly to model disks

File Types

- **Regular Files**:
- **Directory**:
- **Character special files:**
- **Block special files:** (a block device file)
  - Mainly to model disks

- Other files are binary, which just means that they are not ASCII files.
- Usually, they have some internal structure known to programs that use them.

File Operations

- Files exist to store information and allow it to be retrieved later.
- Different systems provide different operations to allow storage and retrieval.
- Below are the most common system calls relating to files.

  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Append
  - Seek
  - Get Attributes
  - Set Attributes
  - Rename

Access Methods

- Files store information and this information must be accessed and read into computer memory.
- The information in the file can be accessed in several ways.
- Some systems provide only one access method for files.
- Other systems, such as IBM’s, support many access methods, and choosing the right one for a particular application is a major design problem.

Access Methods

Sequential Access

- The simplest access method is **sequential access**.
- Information in the file is processed in order, one record after the other.
- This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

Access Methods

Sequential Access

- Reads and writes make up the bulk of the operations on a file.

- A read operation—*read next*—reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.

- Similarly, the write operation—*write next*—appends to the end of the file and advances to the end of the newly written material (the new end of file).

Access Methods

Sequential Access

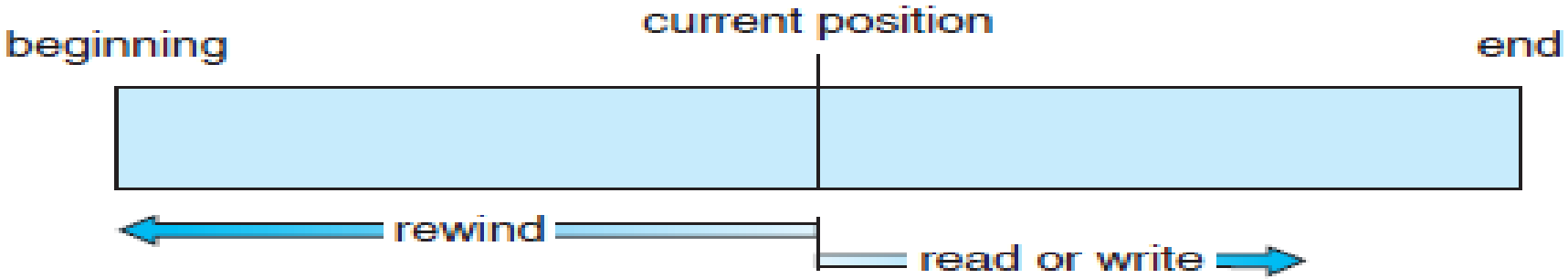


Figure: Sequential-access file Ref: Operating System Concept by Galvin

Access Methods

Direct Access (Relative access)

- A file is made up of fixed length **logical records** that allow programs to read and write records rapidly in no particular order.
- The direct-access method is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on the order of reading or writing for a direct-access file.

Access Methods

Direct Access (Relative access)

- Direct-access files are of great use for immediate access to large amounts of information.
- Databases are often of this type.
- For the direct-access method, the file operations must be modified to include the block number as a parameter.
- Thus, we have *read n*, where *n* is the block number, rather than *read next*, and *write n* rather than *write next*.

Access Methods

Other Access Methods

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.
- The **index**, like an index in the back of a book, contains pointers to the various blocks.
- To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.

Access Methods

Other Access Methods

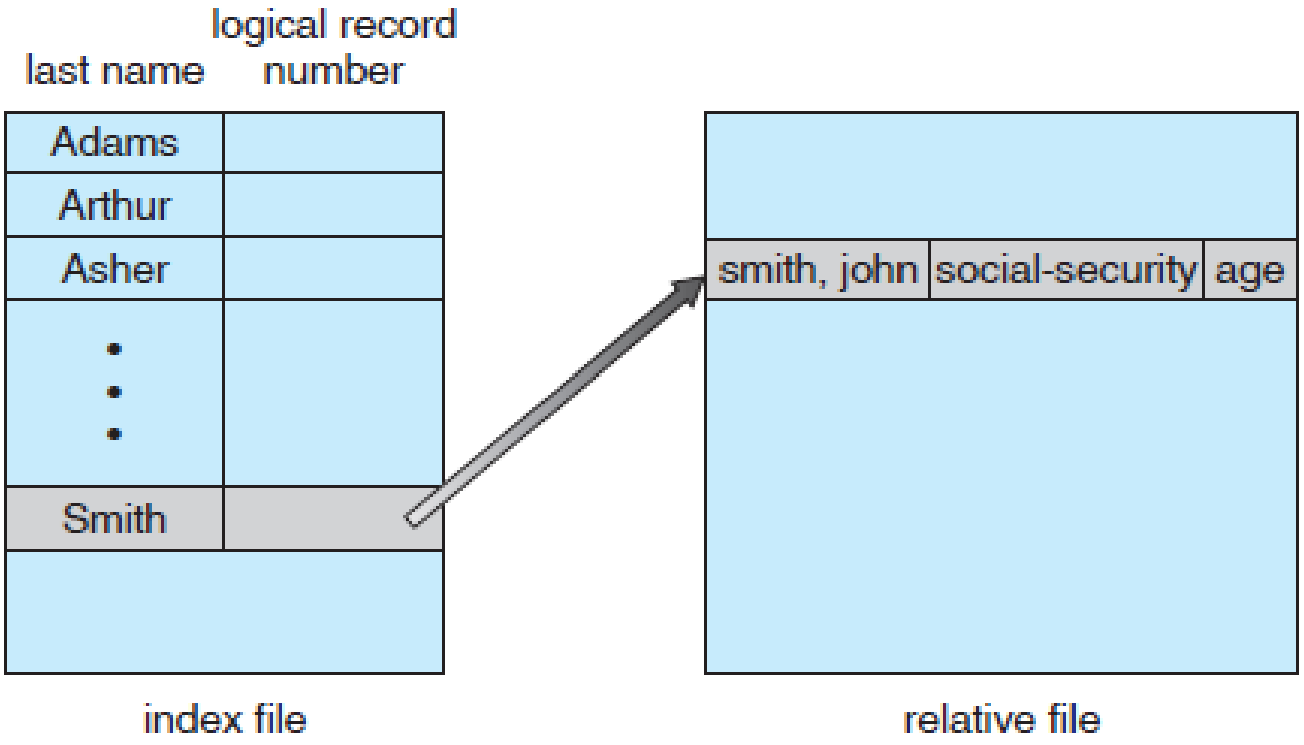


Fig.: Example of index and relative files, Ref: Operating System Concept by Galvin

Directories

- To keep track of files, file systems normally have directories or folders, which are themselves files.
- Two directory systems
  - Single-Level Directory Systems
  - Hierarchical Directory Systems

Directories

Single-Level Directory Systems

- The simplest form of directory system is having one directory containing all the files.
- Sometimes it is called the **root directory**, but since it is the only one, the name does not matter much.
- On early personal computers, this system was common, in part because there was only one user.

Directories

Single-Level Directory Systems

- The world's first supercomputer, the CDC 6600, also had only a single directory for all
- This system was to keep the software design simple.
- The advantages of this scheme are its simplicity and the ability to locate files quickly.
- It is still used on simple embedded devices such as digital cameras and some portable music players.

Directories

Single-Level Directory Systems

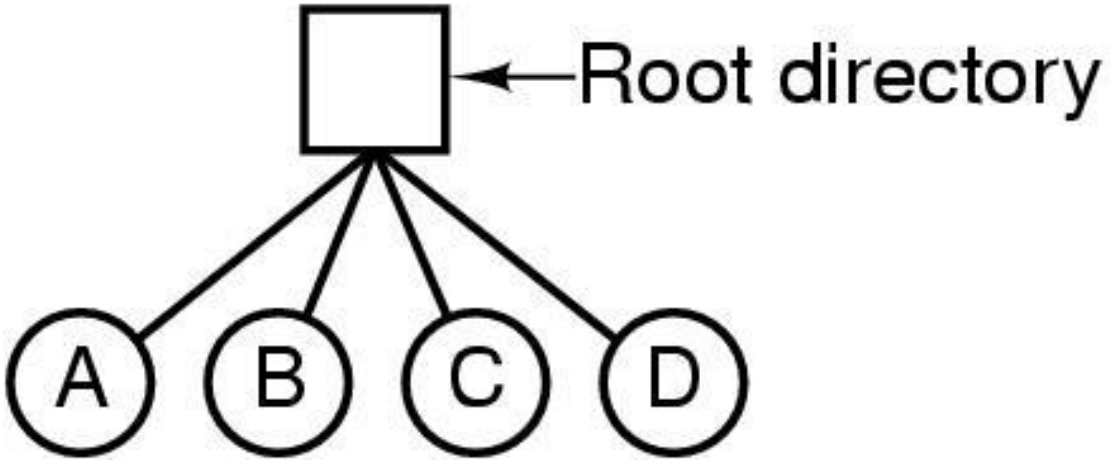


Fig.: A single-level directory system containing four files.

Directories

Hierarchical Directory Systems

- Modern users with thousands of files, it would be impossible to find anything if all files were in a single directory.

- Consequently, a way is needed to group related files together.

- With this approach, there can be as many directories as are needed to group the files in natural ways.

Directories

Hierarchical Directory Systems

- Furthermore, if multiple users share a common file server, as is the case on many company networks, each user can have a private root directory for his or her own hierarchy.

Directories

Hierarchical Directory Systems

- This approach is shown in Fig.

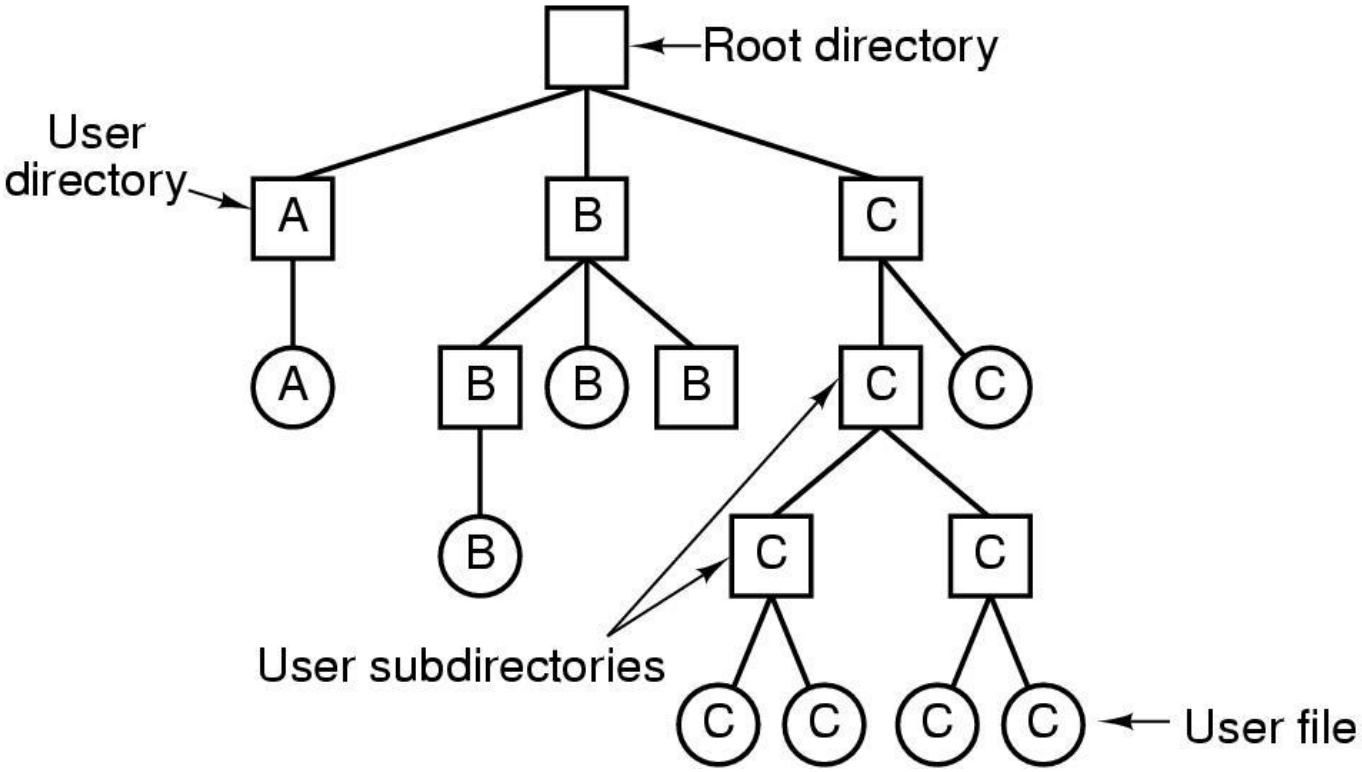


Fig.: A hierarchical directory system.

Directories

Hierarchical Directory Systems

- Here, the directories *A*, *B*, and *C* contained in the root directory each belong to a different user, two of whom have created subdirectories for projects they are working on.
- Nearly all modern file systems are organized in this manner.

Directories

Hierarchical Directory Systems

Path Names

- When the file system is organized as a directory tree, some way is needed for specifying file names.
- Two different methods are commonly used.
    1. Absolute path name
    2. Relative path name

Directories

Hierarchical Directory Systems

Absolute path name:

- **C**onsisting of the path from the root directory to the file.
- As an example, the path */usr/ast/mailbox* means that the root directory contains a subdirectory *usr*, which in turn contains a subdirectory *ast*, which contains the file *mailbox*.

Directories

Hierarchical Directory Systems

Absolute path name:

- Absolute path names always start at the root directory and are unique.
- In UNIX the components of the path are separated by /.
- In Windows the separator is \ .
- In MULTICS it was >.
- Thus, the same path name would be written as follows in these three systems:
  - ✓ Windows \usr\ast\mailbox
  - ✓ UNIX /usr/ast/mailbox
  - ✓ MULTICS >usr>ast>mailbox

Directories

Hierarchical Directory Systems

Relative path name:

- This is used in conjunction with the concept of the **working directory**.
- A user can designate one directory as the current working directory, in which case all path names are taken relative to the working directory (not beginning at the root directory).
- For example, if the current working directory is */usr/ast*, then the file whose absolute path is */usr/ast/mailbox* can be referenced simply as *mailbox*.

Directories

Path Names

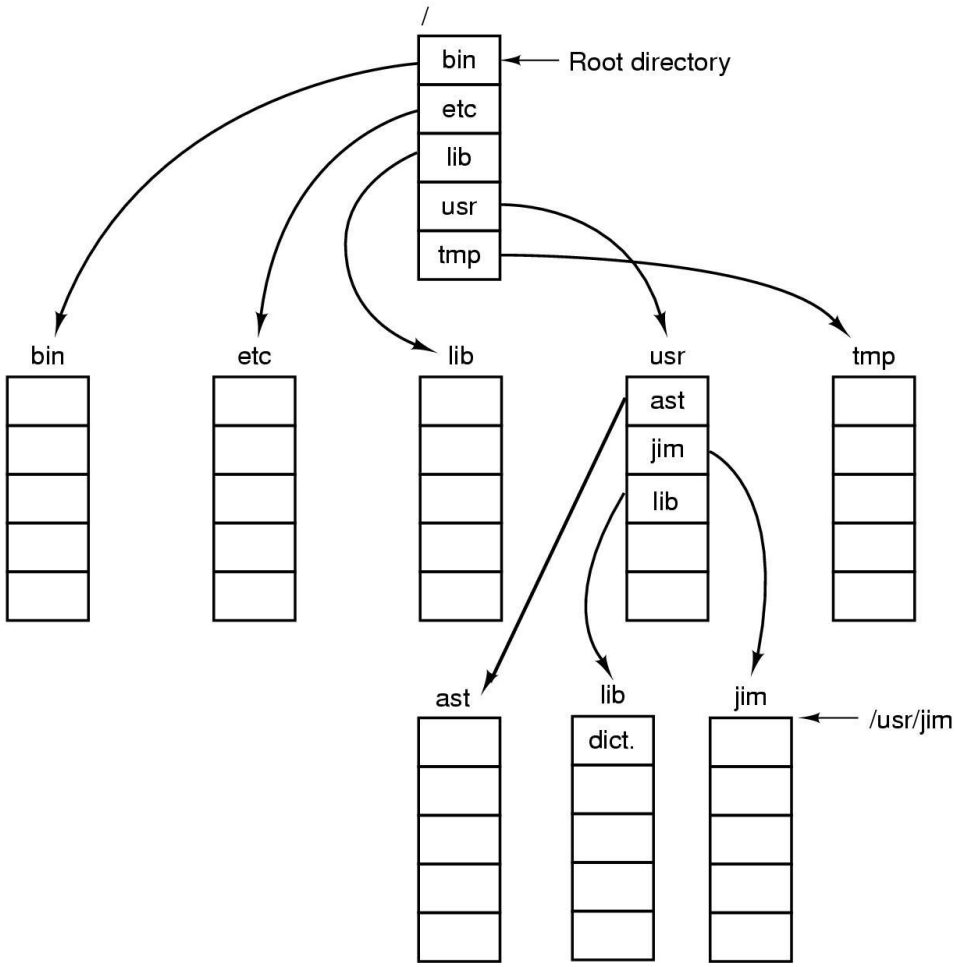


Fig.: A UNIX directory tree.

Directory Operations

System calls for managing directories:

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

File System Implementation

- Users:

  –How files are names, what operations are allowed on them, what the directory tree looks like

- Implementors

  –How files and directories are stored, how disk space is managed and how to make everything work efficiently and reliably

File System Implementation

- Several structures are used to implement a file system.
- These structures vary depending on the operating system and the file system, but some general principles apply.
- The file system may contain information about how to boot an OS stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.
- We briefly describe these structures here;

File System Implementation

Boot-control block:

- A boot control block can contain information needed by the system to boot an operating system from that volume.

File System Implementation

Boot-control block:

- A boot control block can contain information needed by the system to boot an operating system from that volume.

- If the disk does not contain an operating system, this block can be empty.

- It is typically the first block of a every volume.

File System Implementation

Boot-control block:

- A boot control block can contain information needed by the system to boot an operating system from that volume.

- If the disk does not contain an operating system, this block can be empty.

- It is typically the first block of a every volume.

✓In UFS, it is called the **boot block**;
✓In NTFS, it is the **partition boot sector**.

File System Implementation

Volume-control block

- A volume control block contains Volume or partition details, such as
  - Number of blocks in the partition,
  - Size of the blocks,
  - Free-block count and free-block pointers, and
  - Free-File Control Block (FCB) count and FCB pointers.

File System Implementation

Volume-control block

- A volume control block contains Volume or partition details, such as
  - Number of blocks in the partition,
  - Size of the blocks,
  - Free-block count and free-block pointers, and
  - Free-File Control Block (FCB) count and FCB pointers.

  ✓ In UFS, this is called a **superblock**;
  ✓ In NTFS, it is stored in the **master file table**.

File System Implementation

Directory structure

- Directory is present per file system.
- It is used to organize the files.

File System Implementation

Directory structure

- Directory is present per file system.
- It is used to organize the files.

✓In UFS, this includes file names and associated inode numbers.

✓In NTFS, it is stored in the master file table.

File System Implementation

File Control Block

- A per-file FCB contains many details about the file.
- It has a unique identifier number to allow association with a directory entry.

File System Implementation

File Control Block

- A per-file FCB contains many details about the file.
- It has a unique identifier number to allow association with a directory entry.

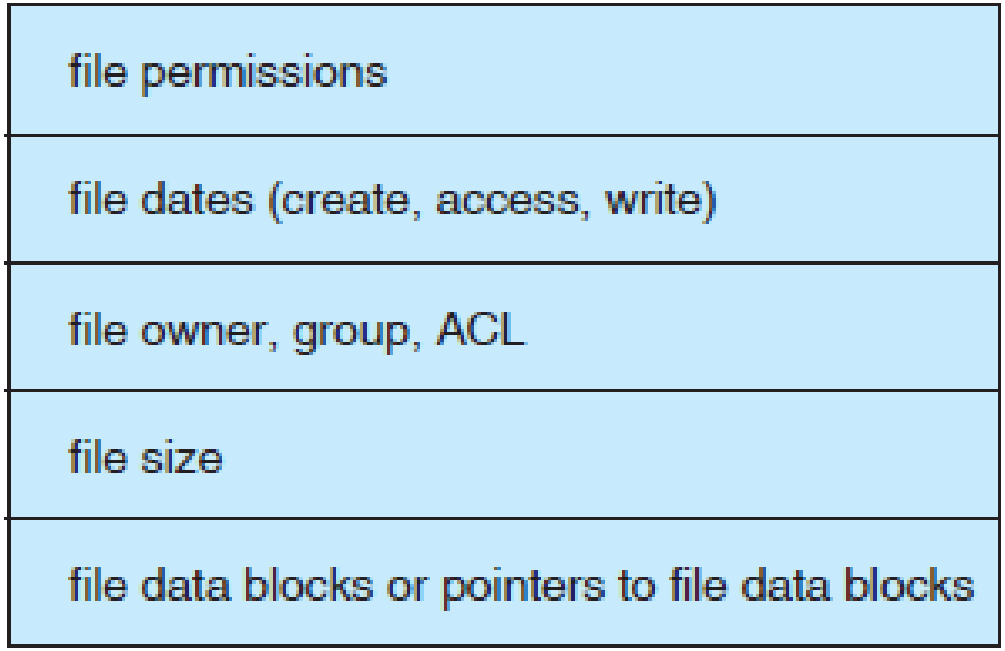


Fig.: A typical file-control block.

File System Implementation

File Control Block

- A per-file FCB contains many details about the file.
- It has a unique identifier number to allow association with a directory entry.

✓In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file.

File System Layout

- File system are stored on disks.
- Most disks are divided up into several partitions
- Sector 0 is called MBR (master boot record), to boot the computer.
- BIOS reads in and executes MBR.
- MBR locates the active partition, reads in the boot block, and execute.
- The boot block reads in the OS contained in the partition.

File System Layout

- Superblock: contains all the key parameters about a file system; read into memory the booted or the FS is used

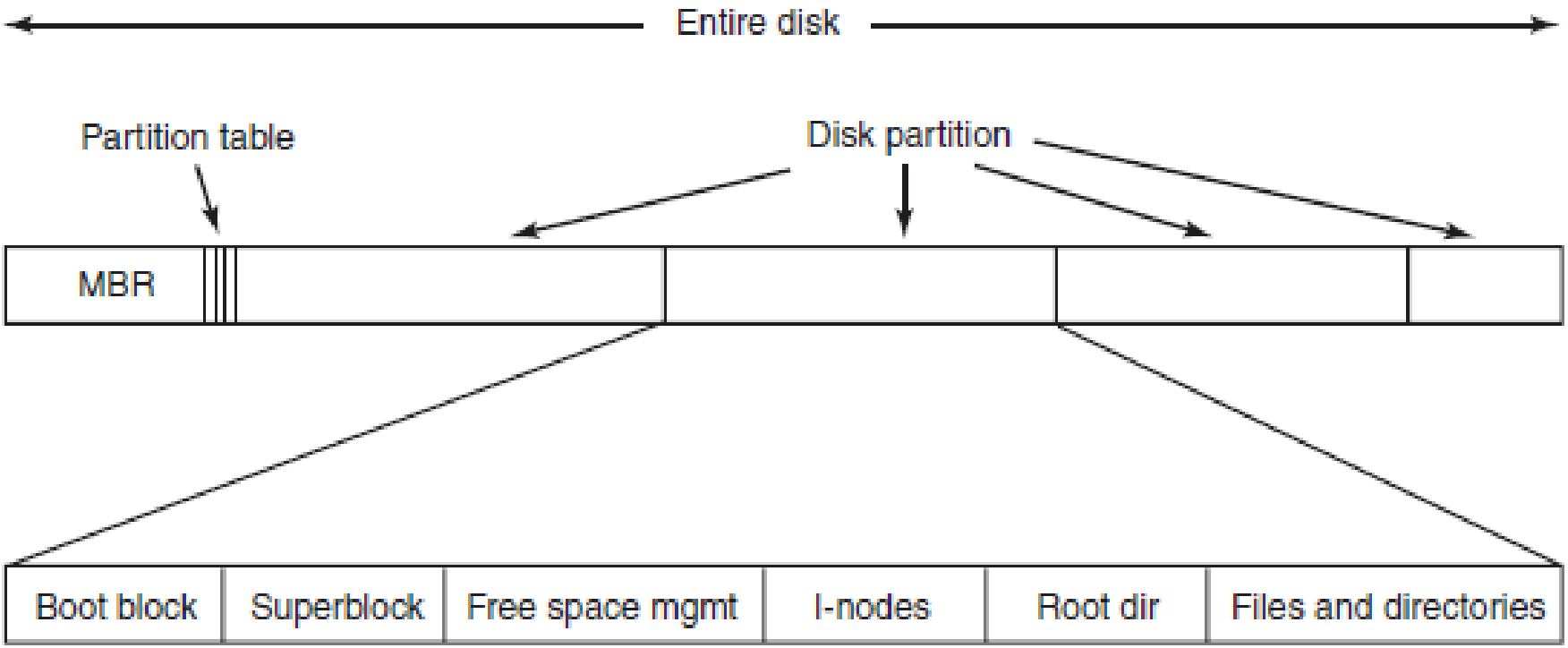


Fig.: A possible file system layout.

Allocation Methods

- The direct-access nature of disks allows us flexibility in the implementation of files.
- In almost every case, many files are stored on the same disk.
- The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.

Allocation Methods

- Three major methods of allocating disk space are in wide use:
  1. Contiguous
  2. Linked
  3. Indexed

- Each method has advantages and disadvantages.

- Some systems support all three, such as *Data General's RDOS for its Nova line of computers*.

- More commonly, a system uses one method for all files within a file-system type.

Allocation Methods

1. Contiguous Allocation:

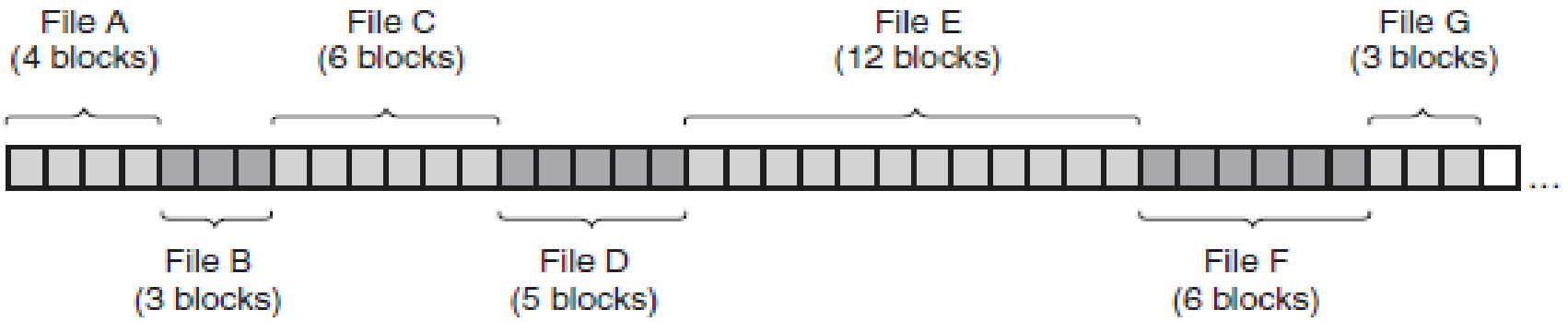
- The simplest allocation scheme is to store each file as a contiguous run of disk blocks.

- Thus on a disk with 1-KB blocks, a 50-KB file would be allocated 50 consecutive blocks.

- With 2-KB blocks, it would be allocated 25 consecutive blocks.

Allocation Methods

1. Contiguous Allocation:



- Here first 40 disk blocks are shown, starting with block 0 on the left.
- Initially, the disk was empty. Then a file *A*, of length four blocks, was written to disk starting at the beginning (block 0).
- After that a six-block file, *B*, was written starting right after the end of file *A*.

Allocation Methods

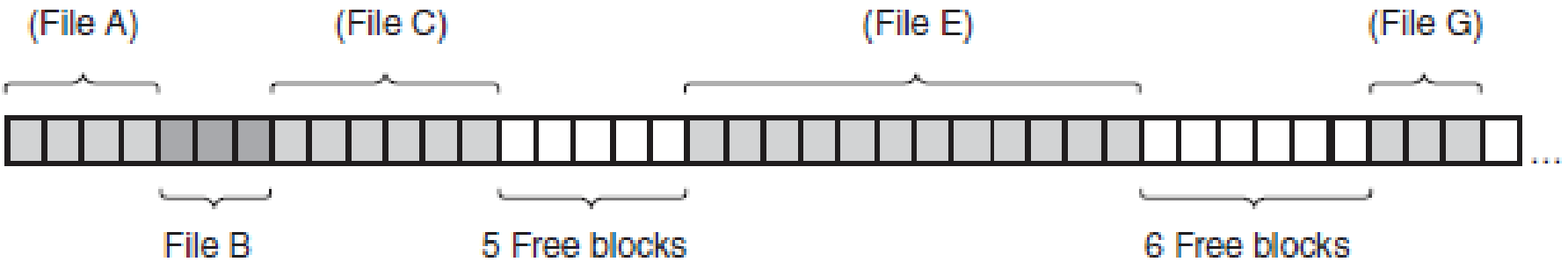
1. Contiguous Allocation:

- Advantage：
  - –Simple to implement; to record the first block and length
  - –Easy to read; only one seek is needed

Allocation Methods

1. Contiguous Allocation:

- Advantage：
  - –Simple to implement; to record the first block and length
  - –Easy to read; only one seek is needed
- Disadvantage:
  - –Fragmentation (see fig. when files D and F removed it create holes.)



Allocation Methods

2. Linked Allocation:

- With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file.

Allocation Methods

2. Linked Allocation:

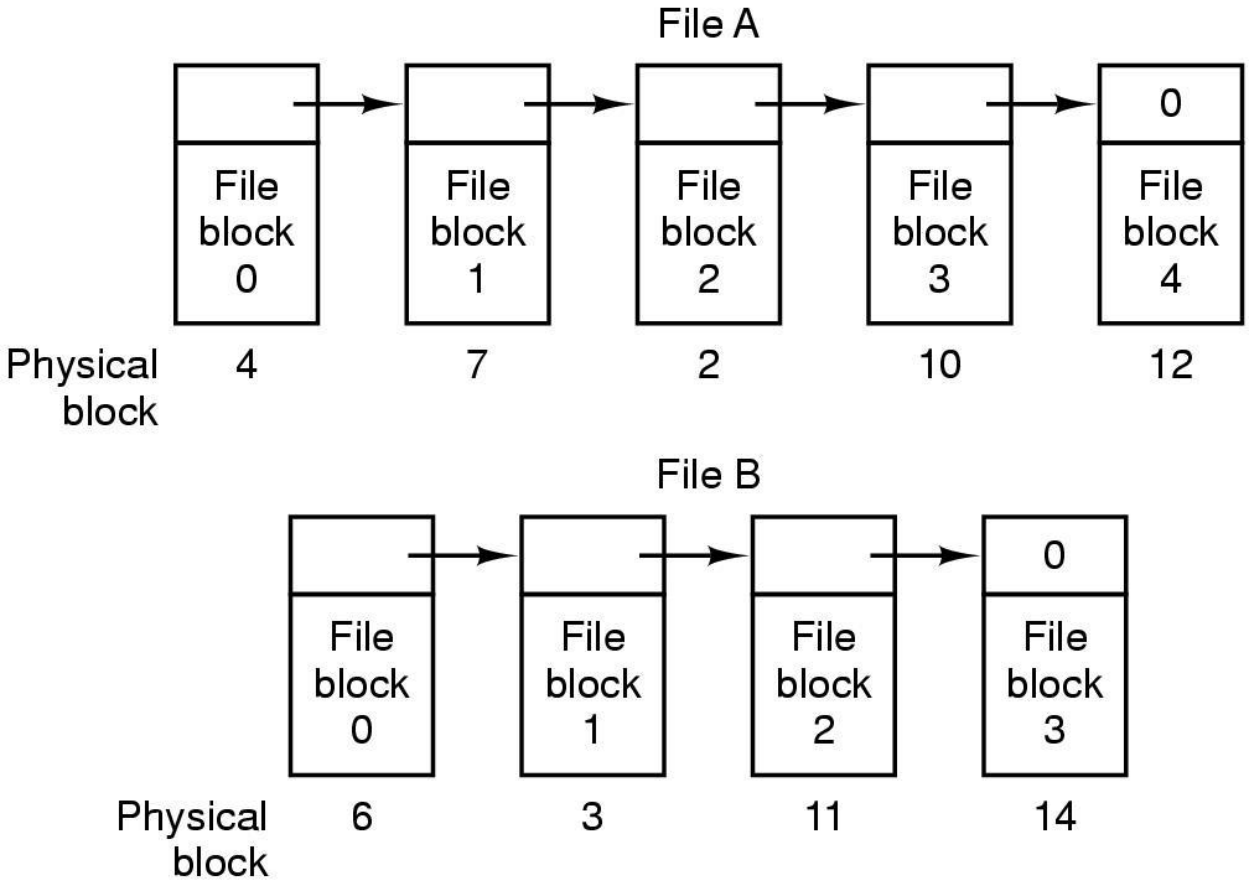


Fig. Storing a file as a linked list of disk blocks.

Allocation Methods

2. Linked Allocation:

- Advantage
  - To create a new file, we simply create a new entry in the directory.
  - There is no external fragmentation with linked allocation.
- Disadvantage
  - Random access is difficult
  - Adding a pointer at the head of block; extra overhead while copying

Allocation Methods

2. Linked Allocation using File Allocation Table:

- Both disadvantages of the linked-list allocation can be eliminated by taking the pointer word from each disk block and putting it in a table in memory.

- Such a table in main memory is called a **FAT**

Allocation Methods

2. Linked Allocation using File Allocation Table:

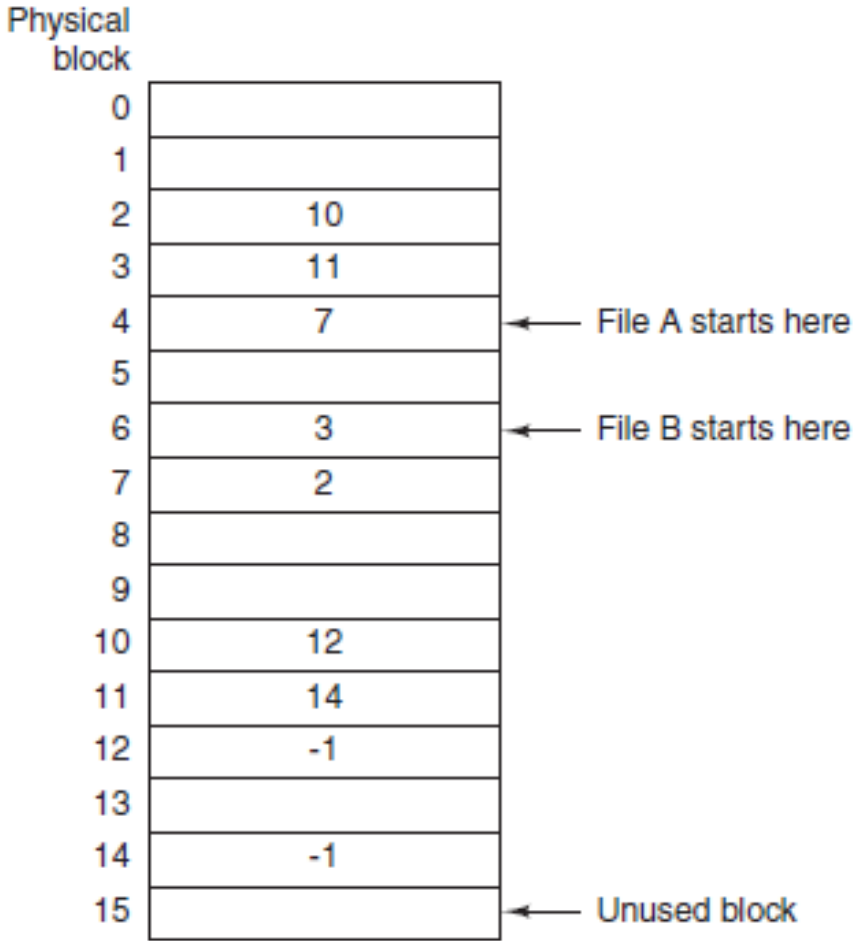


Fig.: Linked list allocation using a file allocation table in main memory.

Allocation Methods

2. Linked Allocation using File Allocation Table:

- Advantage
  - –Can take use of the whole block
  - –only to store the starting block number
- Disadvantage
  - –To keep the entire table in memory
  - –Can’t scale well

Allocation Methods

3. Indexed Allocation

- In the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order.

- **Indexed allocation** solves this problem by bringing all the pointers together into one location: the **index block (index node or i-node)**.

Allocation Methods

3. Indexed Allocation

- Each file has its own i-node, which is an array of disk-block addresses.

- The *ith* entry in the i-node points to the *ith* block of the file.

- The directory contains the address of the i-node.

Allocation Methods

3. Indexed Allocation:

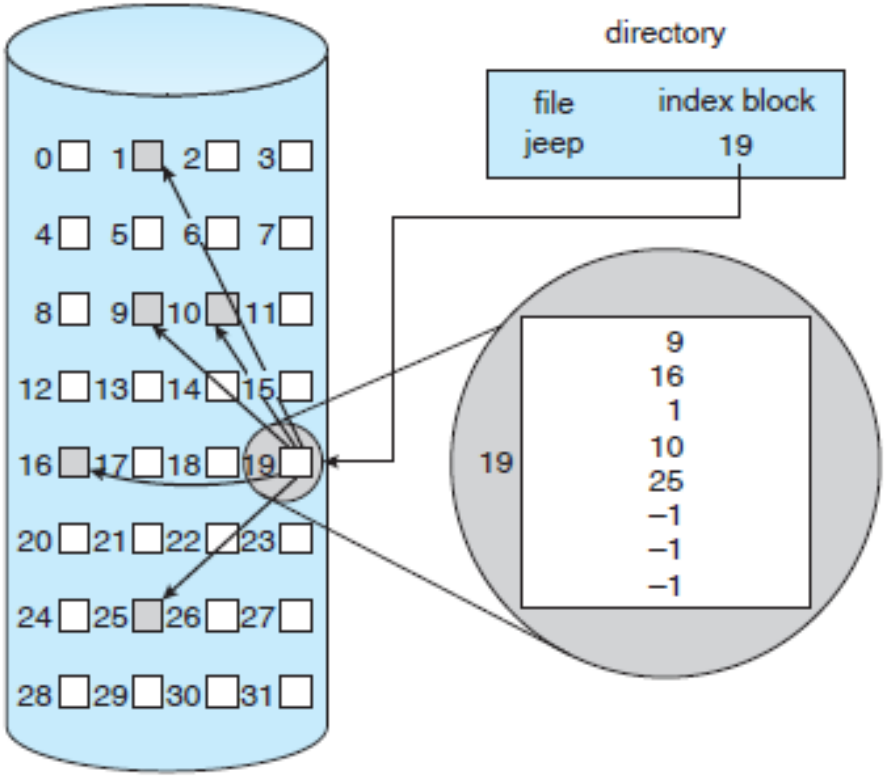


Fig.: Indexed allocation of disk space

Allocation Methods

3. Indexed Allocation

- Advantage
  - Indexed allocation supports direct access without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.
  - i-node need only be in memory when the corresponding file is open; file table grows linearly with the disk
- Disadvantage
  - The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Free Space Management

- Since disk space is limited, we need to reuse the space from deleted files for new files.
- To keep track of free disk space, the system maintains a **free-space list**.

- The free-space list records all *free* disk blocks—those not allocated to some file or directory.

Free Space Management

- To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. And this space is then removed from the free-space list.

- When a file is deleted, its disk space is added to the free-space list.

Free Space Management

- The free-space list, despite its name, might not be implemented as a list.

- The different methods for free-space management are;
  1. Bit vector
  2. Linked list
  3. Grouping
  4. Counting

Free Space Management

1. Bit Vector

- Frequently, the free-space list is implemented as a **bit map** or **bit vector**.

- Each block is represented by 1 bit.
  - If the block is free, the bit is 1;
  - if the block is allocated, the bit is 0.

Free Space Management

1. Bit Vector

- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated.

Free Space Management

1. Bit Vector

- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated.

- The free-space bit map would be
  001111001111110001100000011100000 ...

Free Space Management

1. Bit Vector

- Advantage
  - The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or *n* consecutive free blocks on the disk.

- Disadvantage
  - Bit vectors are inefficient unless the entire vector is kept in main memory

Free Space Management

2. Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.

- This first block contains a pointer to the next free disk block, and so on.

Free Space Management

2. Linked List

- Earlier example, in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated.

- In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on..

Free Space Management

2. Linked List

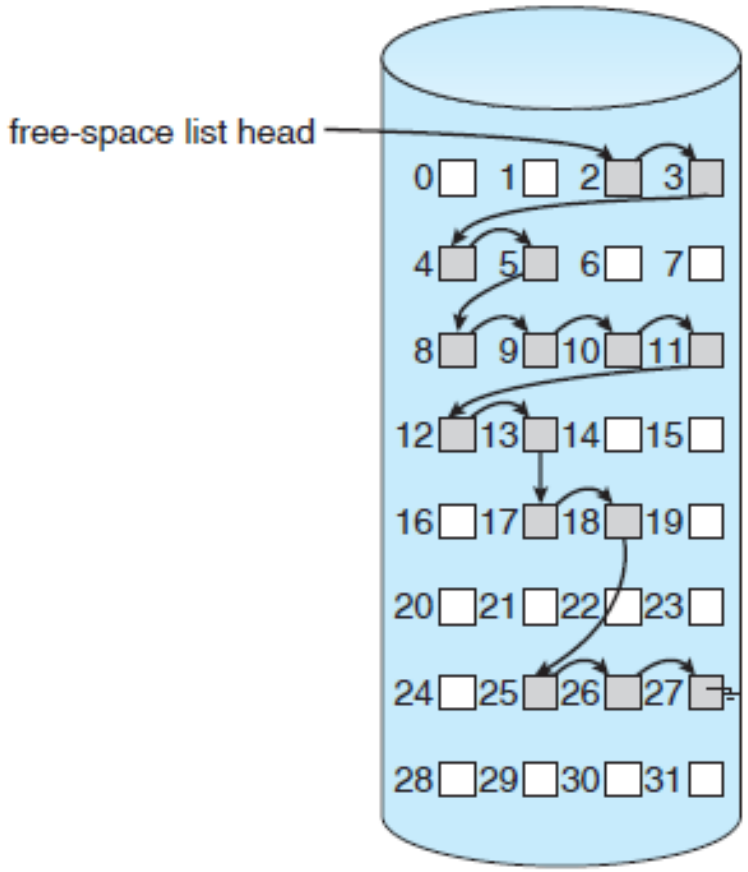


Fig.: Linked free-space list on disk

Free Space Management

2. Linked List

- Advantage
  - Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.

- Disadvantage
  - This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.

Free Space Management

2. Linked List

- Advantage
  - Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.

- Disadvantage
  - This scheme is not efficient; to traverse thelist, we must read each block, which requires substantial I/O time.

- *The FAT method incorporates free-block accounting into the allocation data structure. No separate method is needed.*

Free Space Management

3. Grouping

- A modification of the free-list approach stores the addresses of *n* free blocks in the first free block.
- The first $n-1$ of these blocks are actually free.
- The last block contains the addresses of another *n* free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.

Free Space Management

4. Counting

- This approach takes advantage of several *contiguous* blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.
- Thus, rather than keeping a list of *n* free disk addresses, we can keep the address of the first free block and the number (*n*) of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.

Thank you