Operating System

Chapter 12. File Management

Lynn Choi

School of Electrical Engineering

Files

- **In most applications, files are key elements**
  - For most systems except some real-time systems, files are used as inputs and outputs
  - Virtually all the operating systems provide file systems
- **Desirable properties of files:**
  - Long-term existence
    - Files are stored on disk or other secondary storage
  - Sharable between processes
    - Files have names and can have associated access permissions that permit controlled sharing
  - Structure
    - A file can have an internal structure tailored for a particular application. Also, files can be organized into hierarchical structure to reflect the relationships among files

File System

- **File system**
  - Provide a means to store data organized as files and it also provides a collection of functions that can be performed on files
  - Typical operations include
    - Create, delete, open, close, read, write
  - Maintain a set of attributes associated with the file
    - Owner, creation time, time last modified, access privileges, and so on.
- **File structure**
  - Four terms are commonly used
    - Field
    - Record
    - File
    - Database

Structure Terms

- **Field**
  - Basic element of data
  - Contain a single value, such as name, date
  - Characterized by length and data type
- **Record**
  - A collection of related fields that can be treated as a unit by an application program
  - Example: employee record contains name, social security number, job, date of hire, etc.
- **File**
  - A collection of similar records
  - Treated as a single entity by users and applications
  - Maybe referenced by name
  - Access control restrictions usually apply at the file level
- **Database**
  - A collection of related files (tables)
    - It may contain all the information related to an organization or project
    - Consist of one or more types of files
  - Relationships among elements of data are explicit
  - Designed for use by a number of different applications
  - Managed by a database management system (DBMS), which is independent of OS

File System Objectives

- **File system**
  - A set of system software that provides service to users and applications in the use of files
  - Typically, the only way that a user or application may access file is through the file system
    - Relieve the user of developing special-purpose software for each application
    - Provide the system with consistent, well-defined means of controlling the files
- **File system objectives**
  - Meet the data management needs of the user
  - Guarantee that the data in the file are valid
  - Optimize performance for throughput and response time
  - Provide I/O support for various storage device types
  - Minimize lost or destroyed data
  - Provide a standardized set of I/O interface routines to user processes
  - Provide I/O support for multiple users in the case of multiple-user systems

File System Requirements

- **Each user**
  - Should be able to create, delete, read, write and modify files
  - May have controlled access to other users' files
  - May control what type of accesses are allowed to each file
  - Should be able to restructure the files in a form appropriate to the problem
  - Should be able to move data between files
  - Should be able to back up and recover files in case of damage
  - Should be able to access files by name than by numeric identifier

File System Architecture

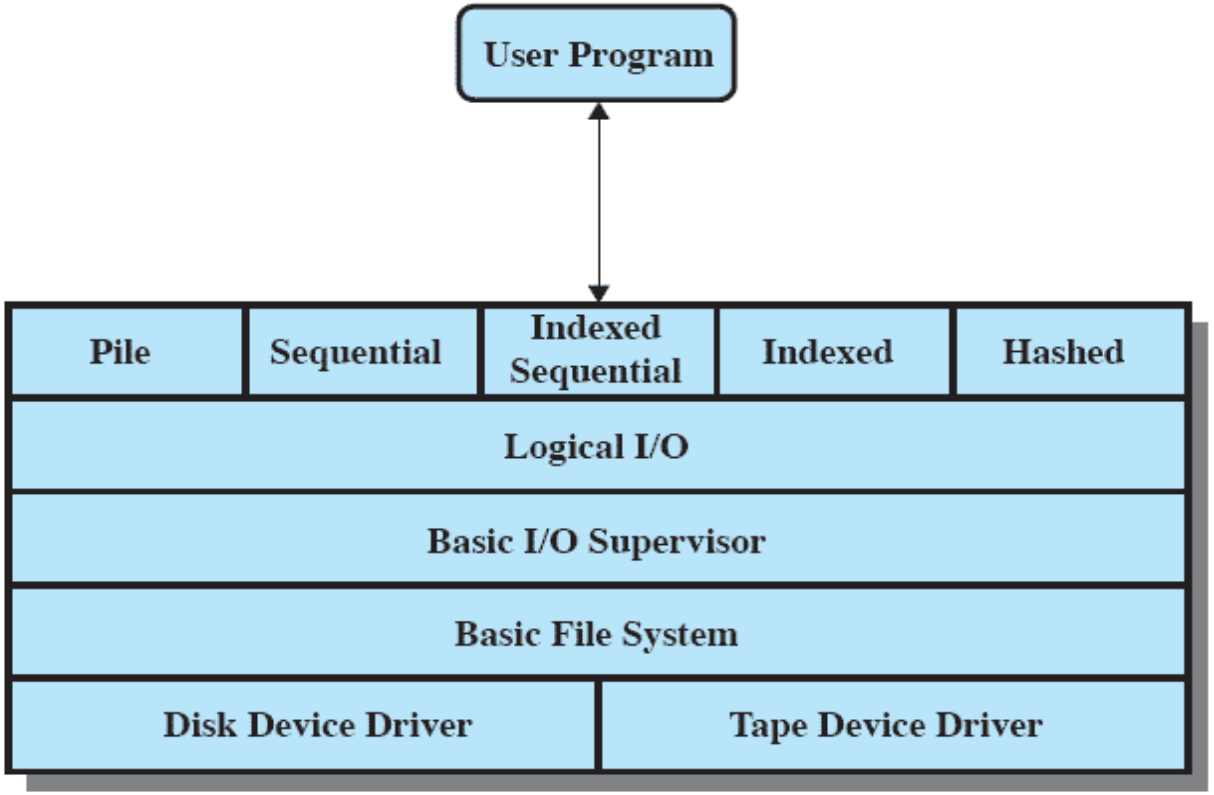


Figure 12.1 File System Software Architecture

Source: Pearson

File System Architecture

- **Device drivers**
  - Lowest level
  - Communicates directly with peripheral devices or their controllers
  - Responsible for starting/completion of I/O operations on a device
  - Part of OS

- **Basic file system**
  - Also referred to as the *physical I/O*
  - Primary interface with the environment outside the computer system
  - Deals with data blocks that are exchanged with disk systems
  - Deals with the placement of blocks on the secondary storage device
  - Deals with buffering blocks in main memory
    - Does not understand the content of data or the structure of the files
  - Part of OS

File System Architecture

- **Basic I/O supervisor**
  - Responsible for file I/O initiation and termination
  - Maintain control structures that deal with device I/O, scheduling, and file status
  - Deals with disk scheduling to optimize performance
  - Assign I/O buffers and allocate secondary memory
  - Part of OS
- **Logical IO**
  - While basic file system deals with blocks, the logical I/O deals with file records
  - Provide general-purpose record I/O capability and maintain basic data about files
- **Access method**
  - Level of the file system closest to the user
  - Provide a standard interface between applications and the file systems
  - Different access methods reflect different file structures and different ways of accessing and processing the data

File System in Different Perspective

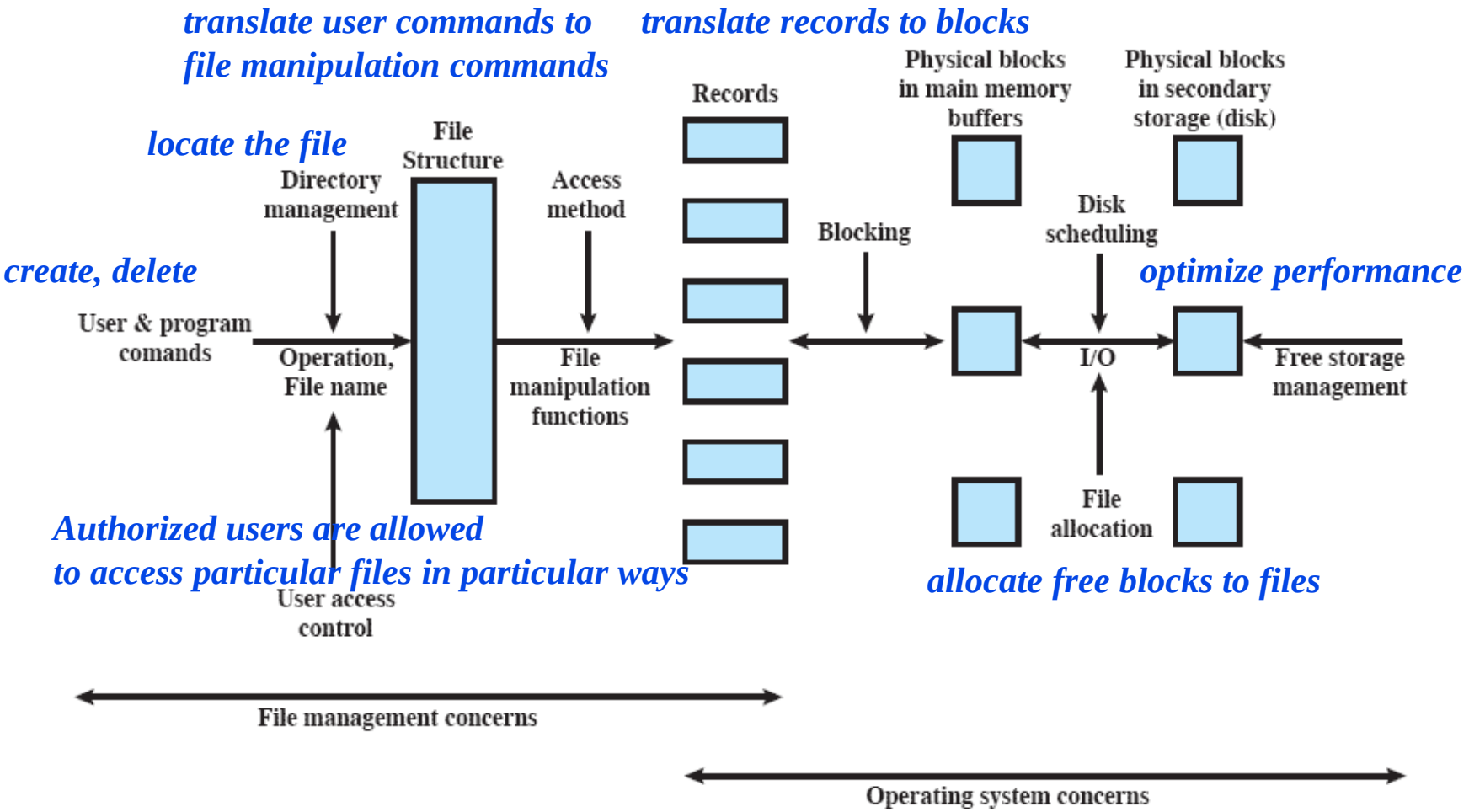


Figure 12.2 Elements of File Management

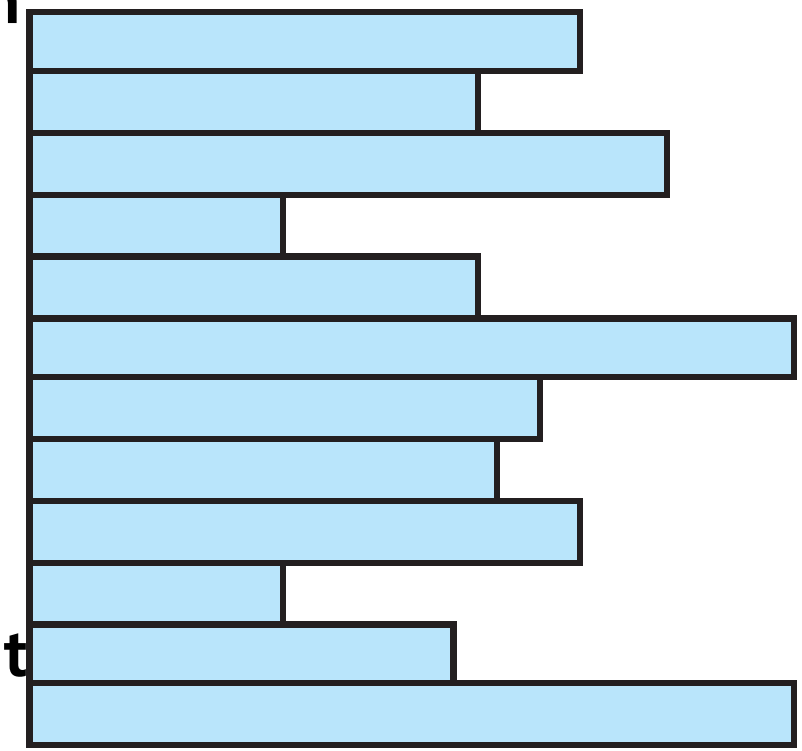
Source: Pearson

File Organization

- **File organization is the *logical structuring* of the records**
  - Physical structuring depends on the blocking and file allocation strategy
- **In choosing a file organization, several criteria are important**
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability
- **Priority of criteria depends on the application**
  - If a file is used only in batch mode, the rapid access of a single record is not important
  - For a file on CD-ROM, the ease of update is not an issue
- **Five common file organization types are**
  - Pile, sequential file, direct or hashed file, indexed file, and indexed sequential file

Pile

- The least complicated form of file organization
  - There is no file structure
- Data are collected in the order in which they arrive
- The purpose is simply to accumulate the mass of data and save it
- Records may have different fields, or similar fields in different order
- Record access is by exhaustive search



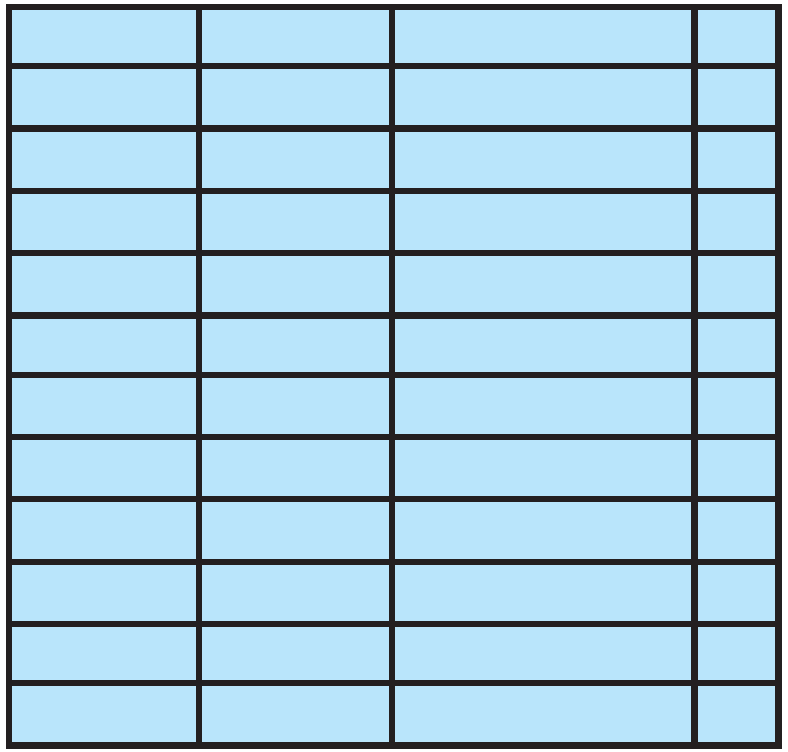
Variable-length records
Variable set of fields
Chronological order

(a) Pile File

Source: Pearson

Sequential File

- The most common form of file structure
- A fixed format is used for records
  - All records have the same length, consisting of the same number of fixed-size fields
- Key field
  - Uniquely identifies the record
  - Records are stored sequentially based on the key field
- Good for batch applications
- Bad for interactive apps.
  - Random access is slow due to sequential search
  - Addition to a file is also slow



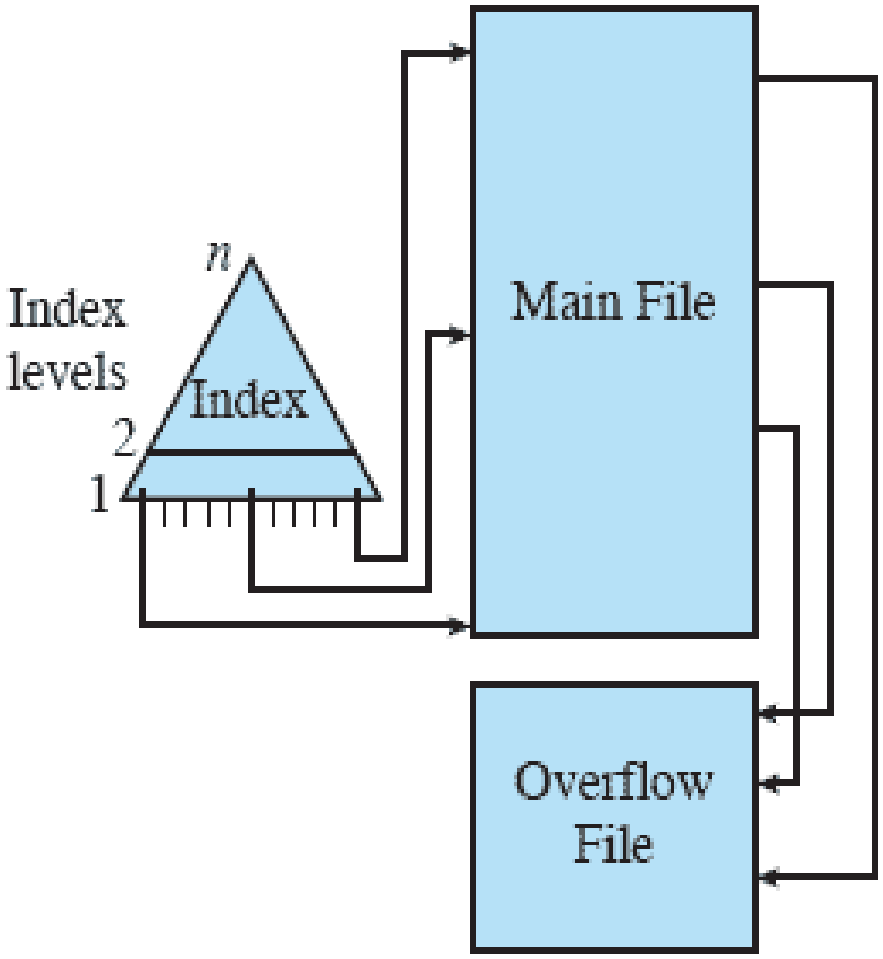
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

Source: Pearson

(b) Sequential File

Indexed Sequential File

- **Two new features**
  - Adds an index to the file to support random access
  - Adds an overflow file to speed up addition
- **Greatly reduces the time required to access a single record**
  - A sequential file with 1M records and1000-entry index requires
    - 500 accesses to the index file + 500 accesses to the main file compared to half million accesses in a sequential file
- **Multiple levels of indexing can be used to provide greater efficiency in access**

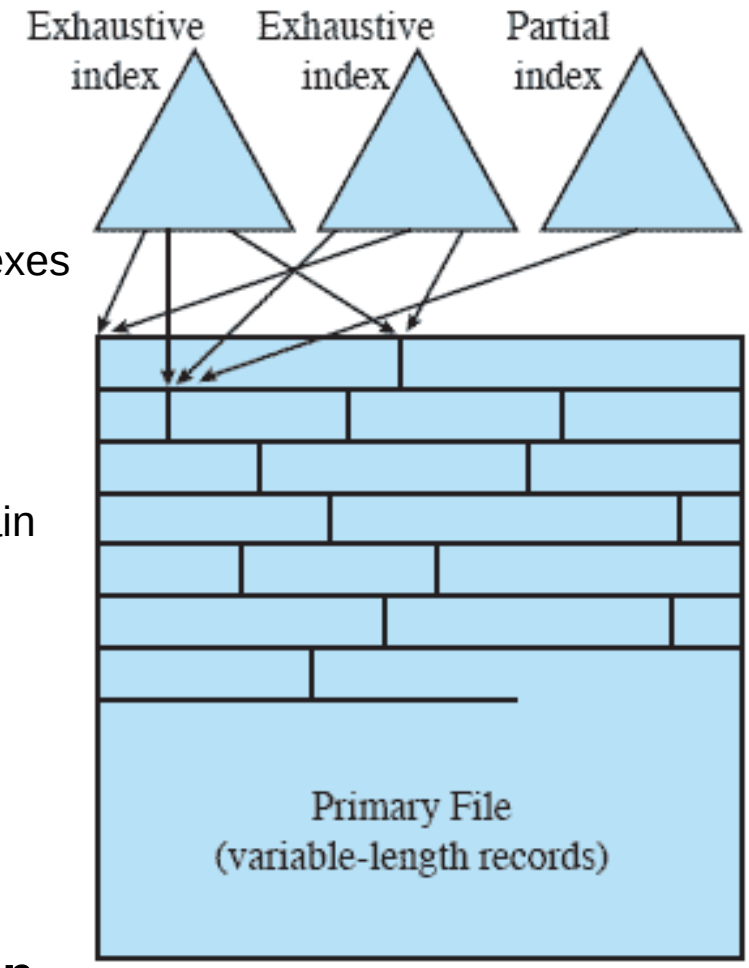


(c) Indexed Sequential File

Source: Pearson

Indexed File

- **Problems of indexed sequential file**
  - Efficient processing is limited to the key field
- **Indexed file**
  - Multiple indexes for each field
  - Records are accessed only through their indexes
    - No restriction on the placement of records
    - Variable-length records can be employed
- **Exhaustive index**
  - Contains one entry for every record in the main file
- **Partial index**
  - Contains entries to records where the field of interest exists
- **Used mostly in applications where timeliness of information is critical**
- **Examples would be airline reservation systems and inventory control systems**



(d) Indexed File

Source: Pearson

Direct or Hashed File

- **Access directly any block of a known address**
- **Makes use of hashing on the key value**
- **Often used where**
  - Very rapid access is required
  - Fixed-length records are used
  - Records are always accessed one at a time
- **Examples are**
  - Directories
  - Pricing tables
  - Schedules

File Directories

- **File directory**
  - Contains information about the files, including attributes, location, and ownership
  - The directory itself is a file
- **Directory operations**
  - Search
    - Search the directory to find an entry for the file
  - Create/delete file
    - Add/delete an entry to the directory
  - List directory
  - Update directory
    - A change in some file attribute requires a change in the directory entry

Basic Information	
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
Address Information	
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
Usage Information	
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

Source: Pearson

Directory Structure

❑ Two-level scheme

- ➤ There is one directory for each user and a master directory
- ➤ Master directory
  - – Has an entry for each user directory
  - – Provide address and access control information
- ➤ User directory
  - – Each user directory is a simple list of the files of that user
  - – Names must be unique only within the collection of files of a single user
- ➤ File system can easily enforce access restriction on directories

❑ Tree-structured directory

- ➤ Master directory with user directories underneath it
- ➤ Each user directory may have subdirectories and files as entries

❑ Each directory can be organized as

- ➤ A sequential file or a hashed file (if the directory contains a very large number of entries

Tree Structured Directory

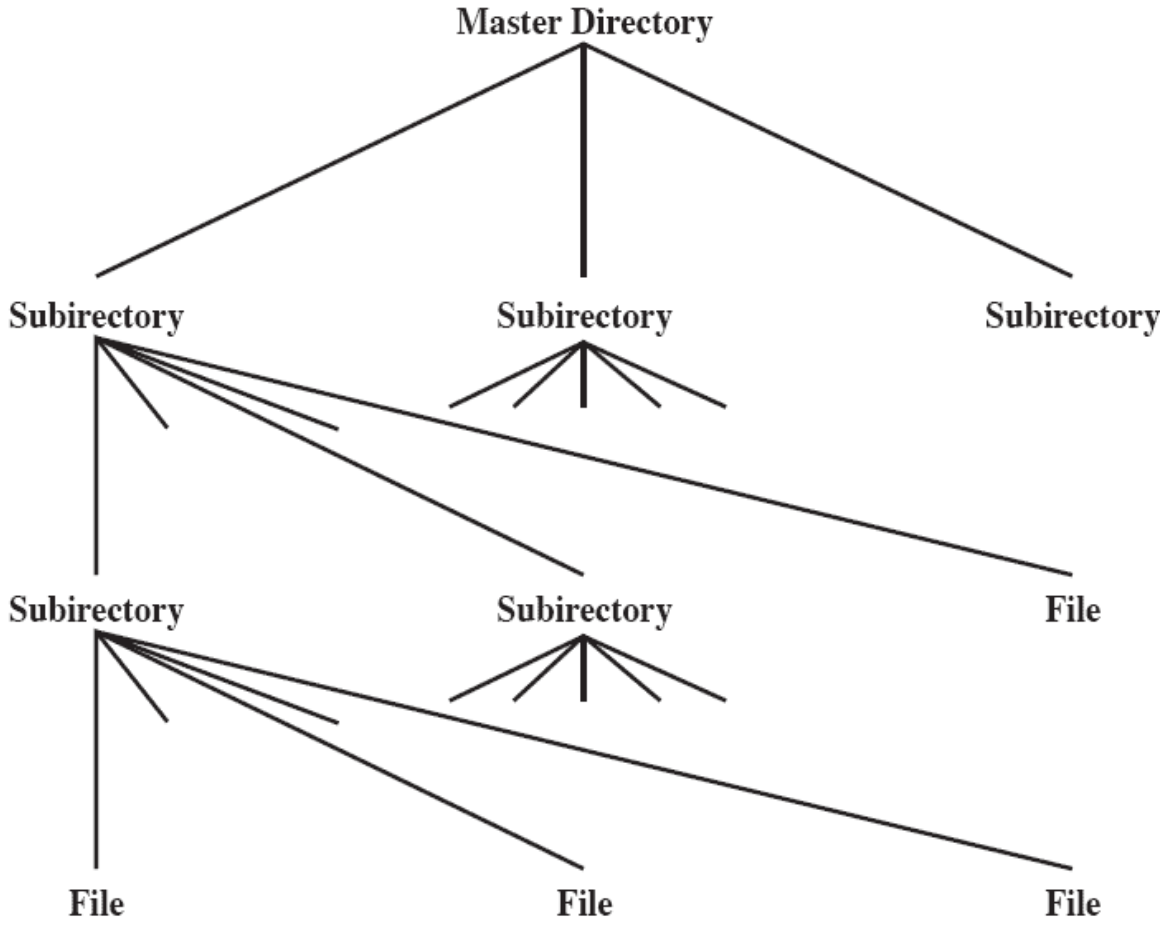


Figure 12.4 Tree-Structured Directory

Source: Pearson

Example of Tree-Structured Directory

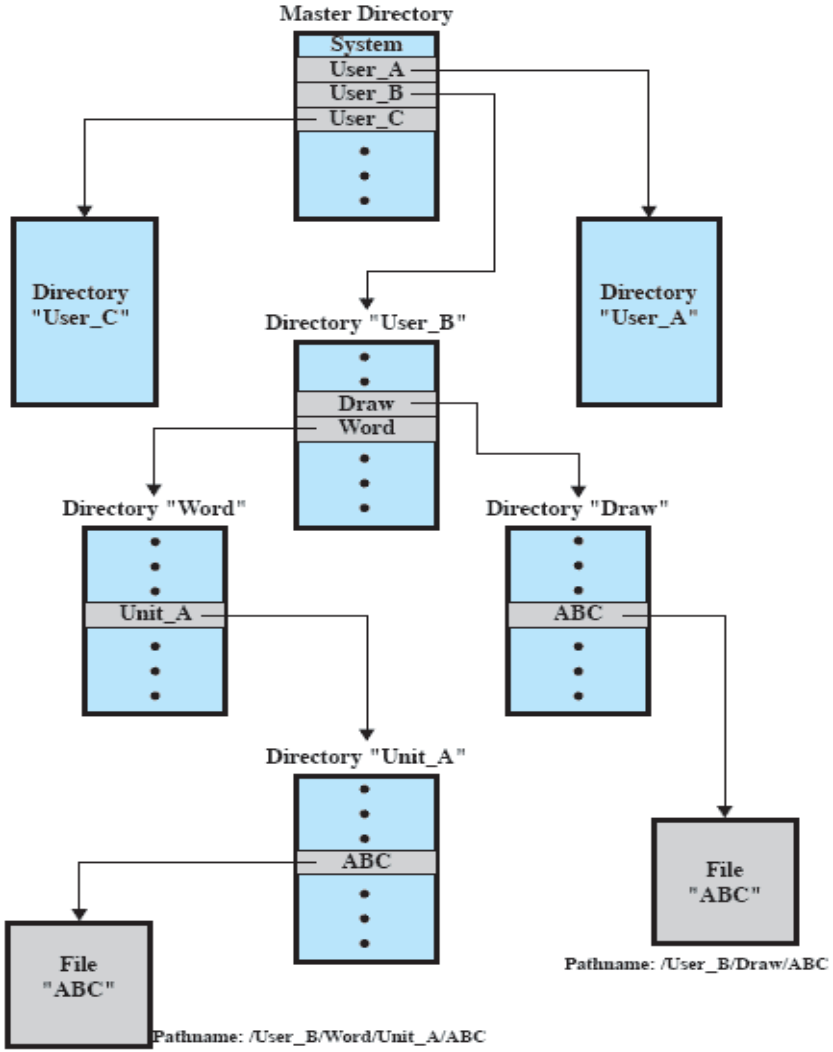


Figure 12.5 Example of Tree-Structured Directory

Source: Pearson

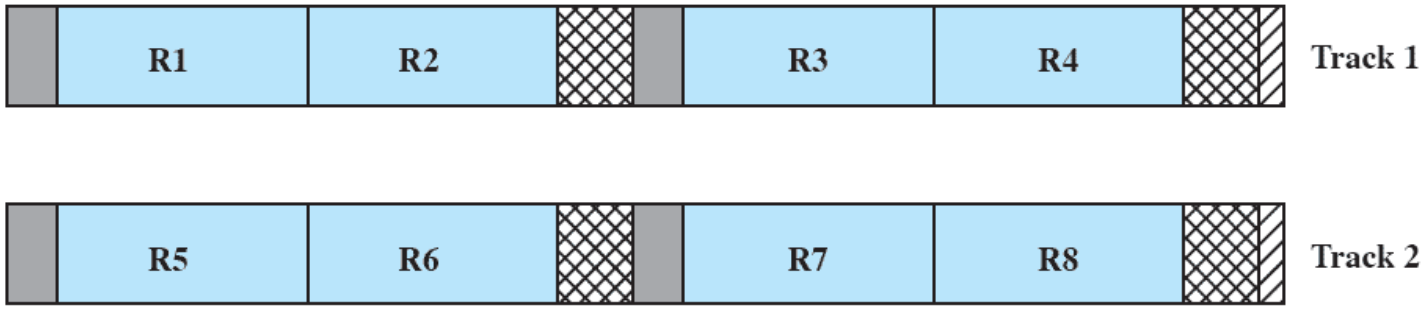
File Sharing

- **Two issues arise**
  - Access rights and the management of simultaneous access (mutual exclusion/deadlock)
- **Access rights**
  - Constitute a hierarchy with each right implying those preceding it
  - None – May not even know the existence of the file. Cannot read the directory
  - Knowledge – Know the file exists and who the owner is. Must ask the owner for access
  - Execute – Can execute the program but cannot copy it
  - Read – Can read the file, including copying and execution
  - Append – Can add data to the file but cannot modify or delete
  - Update – Can modify, delete, and add to the file's data
  - Change protection – Can change the access rights
  - Delete – Can delete the file
- **Access can be provided to different class of users**
  - Owner – The person who initially created the file. Has all the access rights
  - Specific user – Individual user designated by user ID
  - User groups – A group of users
  - All – All users

Record Blocking

- **Records are the logical unit of access for a structured file whereas blocks are the unit of I/O**
  - Thus, to perform I/O, records must be organized as blocks
- **Several issues to consider**
  - Should blocks be of fixed or variable length?
    - In most systems, blocks are of fixed size, which simplify IO
  - What should be the relative size of a block compared to an average record size?
    - The larger the block, can speed up IO but may include unnecessary records
- **Three blocking methods can be used**
  - **Fixed-Length Blocking** – fixed-length records are used, and an integral number of records are stored in a block
  - **Variable-Length Spanned Blocking –** variable-length records are used and are packed into blocks with no unused space
    - Some records may span two blocks
  - **Variable-Length Unspanned Blocking** – variable-length records are used, but spanning is not employed

Fixed Blocking



Fixed Blocking

Data

Gaps due to hardware design

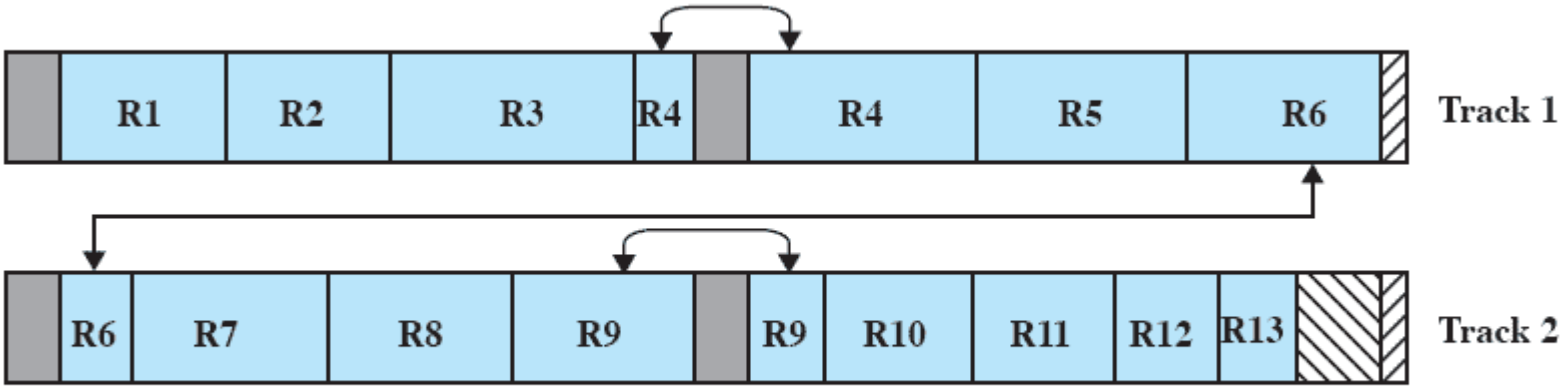
Waste due to block fit to track size

Waste due to record fit to block size

Waste due to block size constraint from fixed record size

Source: Pearson

Variable Blocking: Spanned



Variable Blocking: Spanned

Data

Gaps due to hardware design

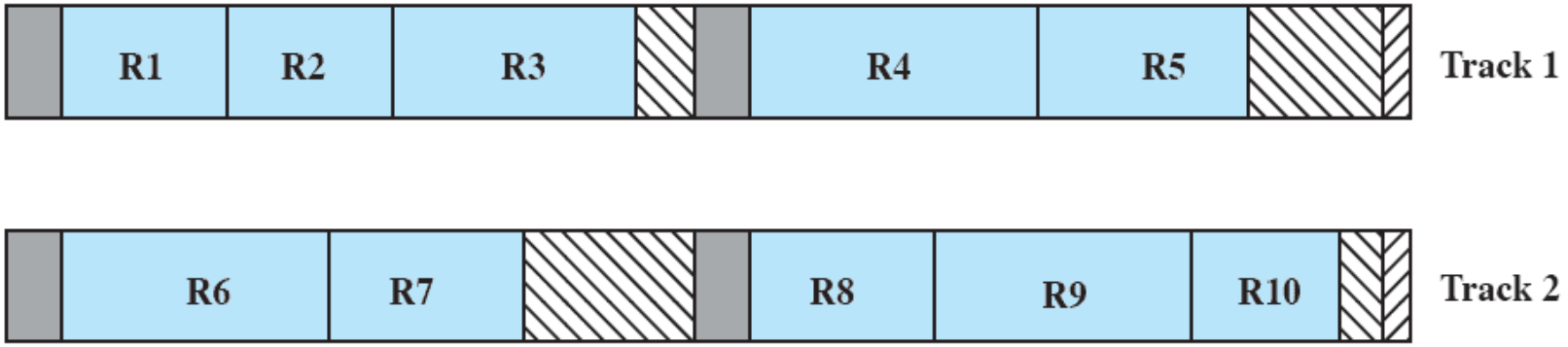
Waste due to block fit to track size

Waste due to record fit to block size

Waste due to block size constraint from fixed record size

Source: Pearson

Variable Blocking: Unspanned



Variable Blocking: Unspanned

- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

Source: Pearson

File Allocation

- **A file consists of a collection of blocks**
- **OS (file system) is responsible for allocating blocks to files**
  - OS allocates free blocks on secondary storage to files
  - OS needs to keep track of free blocks
- **File allocation issues**
  - When a new file is created, should we allocate the maximum space for the file at once?
  - OS assigns a contiguous set of free blocks (called a *portion*) to a file. What size should we use for the portion? It can range from a single block to the entire file.
  - What kind of data structure is used to keep track of the portions assigned to a file?
    - Example: FAT (File Allocation Table) on DOS

Preallocation vs. Dynamic Allocation

- **Preallocation**
  - Require the maximum file size to be declared at the file creation time
  - For some applications, it is possible to estimate the maximum size
    - Program compile, file transfer over the network
  - But, for many applications, it is impossible to estimate the max. size
    - Users and applications tend to overestimate the size, which results in storage waste
- **Dynamic allocation**
  - Allocate space as needed

Portion Size

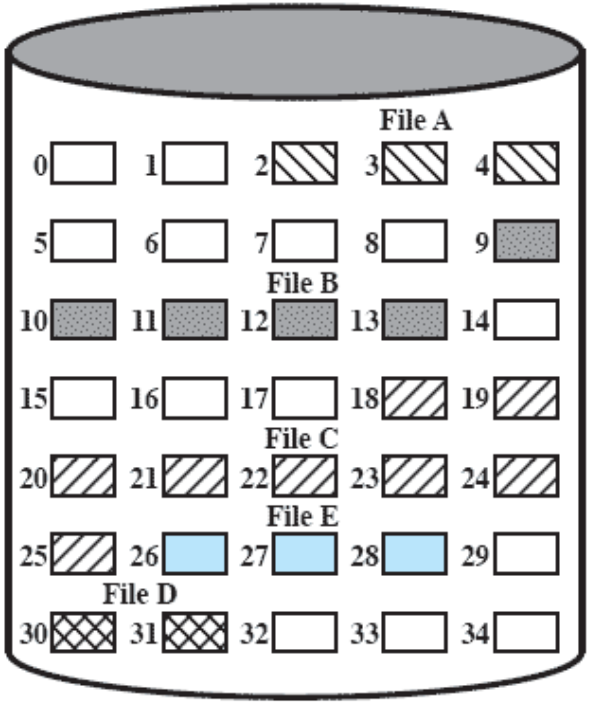
- **The portion size ranges from a single block to the entire file**
- **Need to consider the following:**
  - Contiguity of space increases the performance
  - A large number of small portions increases the size of tables needed to manage the allocation information
  - Fixed size portions simplifies the reallocation of spaces
  - Variable size or small fixed-size portions minimizes the storage waste
- **Two main alternatives**
  - **Variable, large contiguous portions**
    - (+) Better performance, avoid waste, small tables
    - (-) Hard to reuse space (external fragmentation)
  - **Blocks**
    - (+) Greater flexibility, don't have to be contiguous, blocks are allocated on demand, easy to reuse space
    - (-) Large tables

File Allocation Methods

- Contiguous allocation
- Chained allocation
- Indexed allocation

Contiguous File Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation
- Preallocation strategy using variable-size portions
- Advantages
  - Improve I/O performance for sequential processing
  - FAT needs a single entry for each file
- Disadvantages
  - External fragmentation
    - Compaction required



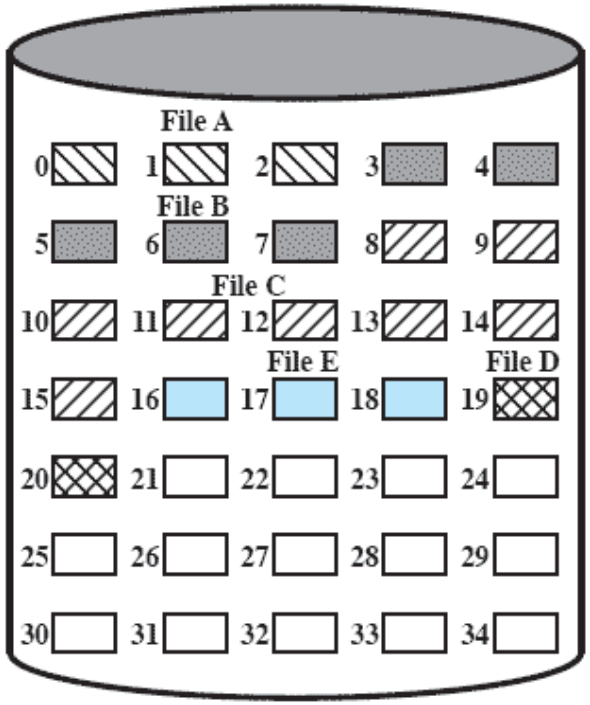
File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.7 Contiguous File Allocation

Source: Pearson

After Compaction



File Allocation Table

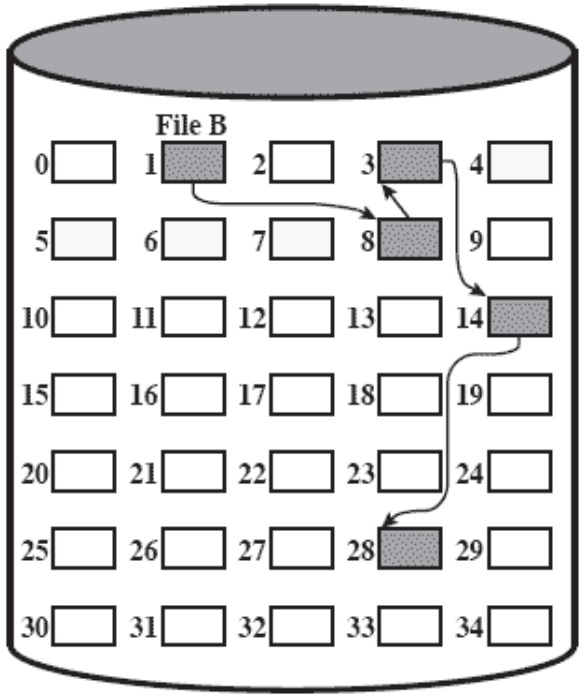
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.8 Contiguous File Allocation (After Compaction)

Source: Pearson

Chained Allocation

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- FAT needs just a single entry for each file
- No external fragmentation
- Not good when we need to bring in multiple blocks since it requires a series of accesses to different parts of disk storage
  - Locality cannot be exploited
  - Require periodic consolidation



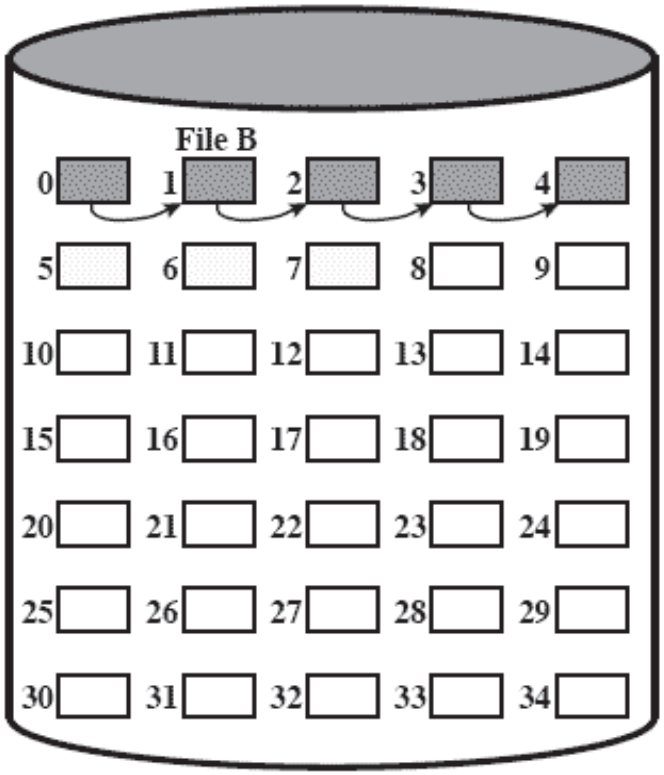
File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Source: Pearson

Figure 12.9 Chained Allocation

Chained Allocation After Consolidation



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.10 Chained Allocation (After Consolidation)

Source: Pearson

Indexed Allocation with Block Portions

- Address the problems of contiguous and chained allocation
- FAT entry for each file points to an *index block*, which has one entry for each portion allocated to the file

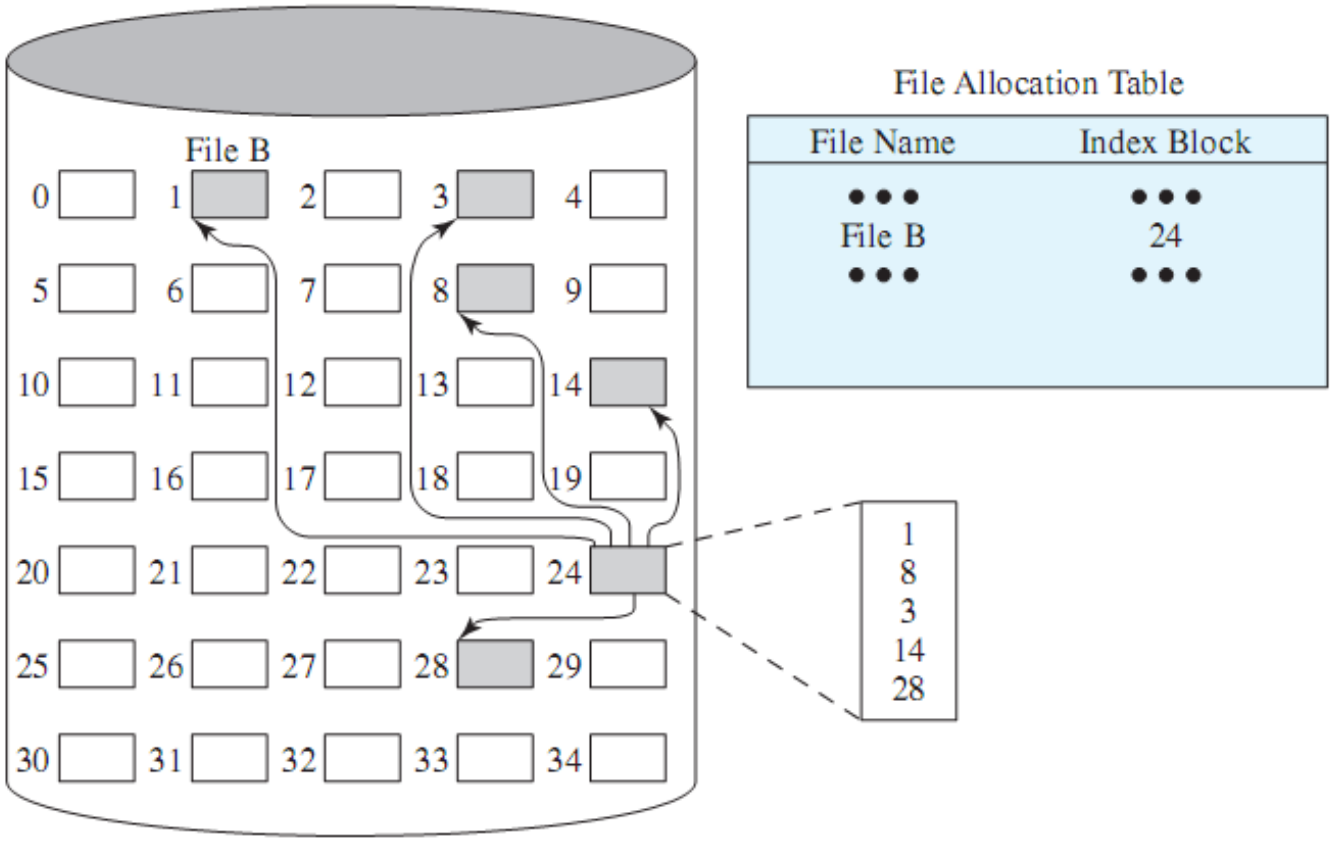


Figure 12.11 Indexed Allocation with Block Portions

Source: Pearson

Indexed Allocation with Variable Length Portions

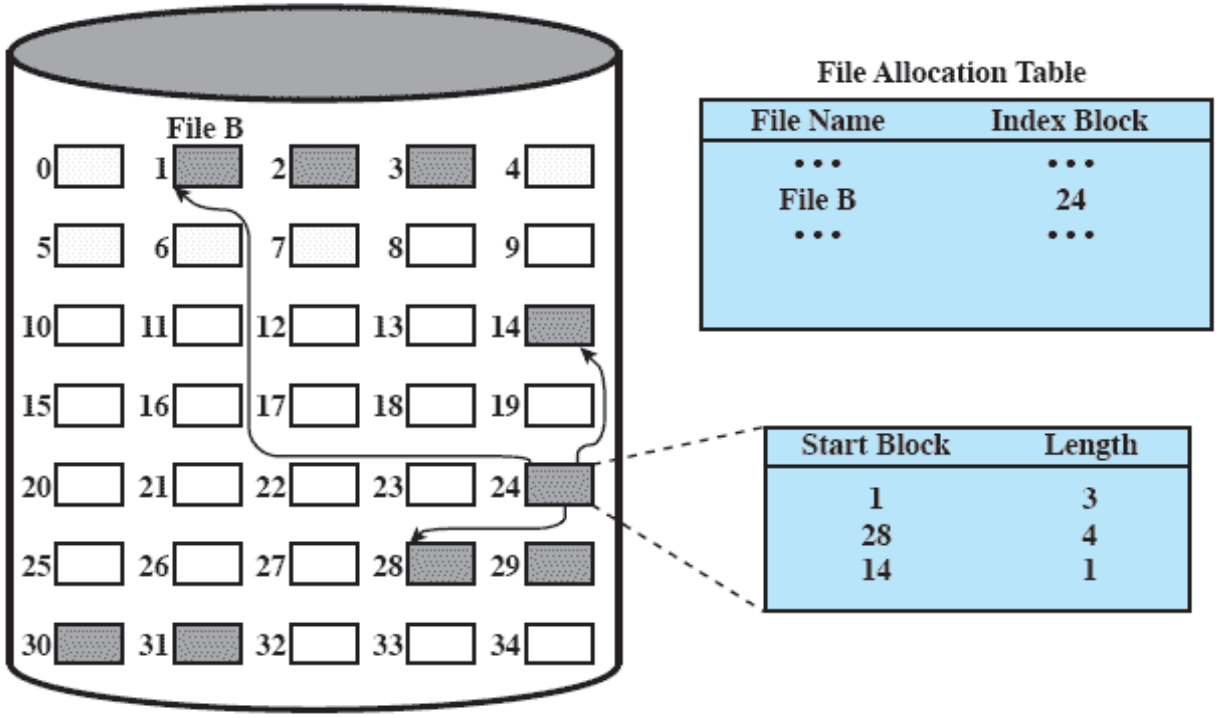


Figure 12.12 Indexed Allocation with Variable-Length Portions

Source: Pearson

Free Space Management

- Just as the allocated space must be managed, so the unallocated space must be managed as well.
- To perform file allocation, it is necessary to know which blocks are available
- A *disk allocation table* is needed in addition to a file allocation table
- Free space management methods
  - Bit table
  - Chained free portions
  - Indexing
  - Free block list

Free Space Management

Bit table

- A bit vector containing one bit for each block on the disk
- 00110000111110000011111101100
- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use
- Works well with any file allocation method
- The size of the bit table is relatively small but can be still big!
  - 16GB disk with 512B blocks needs 4MB bit table, which requires 8000 disk blocks!

Chained free portions

- The free portions may be chained together by using a pointer and length value in each free portion
- Negligible space overhead because there is no need for a disk allocation table
- Suited to all file allocation methods

Free Space Management

- Disadvantages
  - Leads to fragmentation
  - Every time you allocate a block you need to read the block first to recover the pointer to the next free block before writing data to that block
    - If many individual blocks are allocated at once, this greatly slows file creation

❑ Indexing

- Treats free space as a file and uses an index table as it would for file allocation
- For efficiency, the index should be on the basis of variable-size portions rather than blocks
- This approach provides efficient support for all of the file allocation methods

❑ Free block list

- Each block is assigned a number sequentially
- Free block list maintains a list of the numbers of all free blocks
- Assuming 32-bit block number and 512B blocks, less than 1% space overhead
- Effective techniques for storing a small part of the list in main memory
  - LIFO stack with the first few thousand top entries kept in main memory
  - FIFO queue with only the first few thousand head/tails in main memory

Volumes

- **A volume is a logical disk**
  - A collection of addressable sectors in secondary memory that an OS or application can use for data storage
  - The sectors in a volume need not be consecutive on a physical storage device
    - They need only appear that way to the OS or application
  - A volume may be the result of assembling and merging smaller volumes
- **Examples**
  - In the simplest case, a single disk is one volume
  - Frequently, a disk is divided into partitions, with each partition functioning as a separate volume
  - Partitions on multiple disks as a single volume

Homework 11

- Exercise 12.1
- Exercise 12.3
- Exercise 12.6
- Exercise 12.7