Operating Systems: Internals and Design Principles

Chapter 12
File Management

Seventh Edition

By William Stallings

Files

- Data collections created by users
- The File System is one of the most important parts of the OS to a user
- Desirable properties of files:

Long-term existence

- files are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes

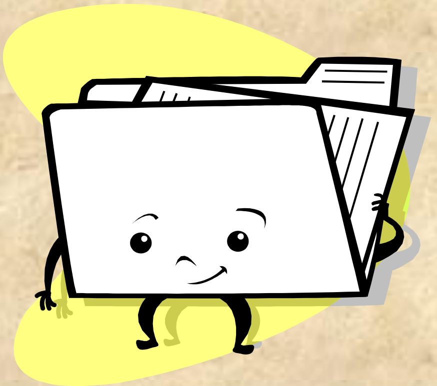
- files have names and can have associated access permissions that permit controlled sharing

Structure

- files can be organized into hierarchical or more complex structure to reflect the relationships among files

File Systems

- The file system gives users an abstraction of the disk
- It provides a way to store data organized as files as well as a collection of functions that can be performed on files
- Maintain a set of attributes associated with the file
- Typical operations include:
  - Create/Delete
  - Open/Close
  - Read/Write



File Management System Objectives

- Meet the data management needs of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types
- Minimize the potential for 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

Minimal User Requirements

- Each user:

1. should be able to create, delete, read, write and modify files
2. may have controlled access to other users' files
3. may control what type of accesses are allowed to the files
4. should be able to restructure the files in a form appropriate to the problem
5. should be able to move data between files
6. should be able to back up and recover files in case of damage
7. should be able to access his or her files by name rather than by numeric identifier

File Directory Information

Table 12.2 Information Elements of a File Directory

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

Operations Performed on a Directory

- To understand the requirements for a file structure, it is helpful to consider the types of operations that may be performed on the directory:

- Search
- Create files
- Delete files
- List directory
- Update directory

Fig. 12.4: Tree-Structured Directory

- Master directory with user directories
- Each user directory may have sub-directories and files as entries
- Simplifies require-ments for unique file names across multiple users.

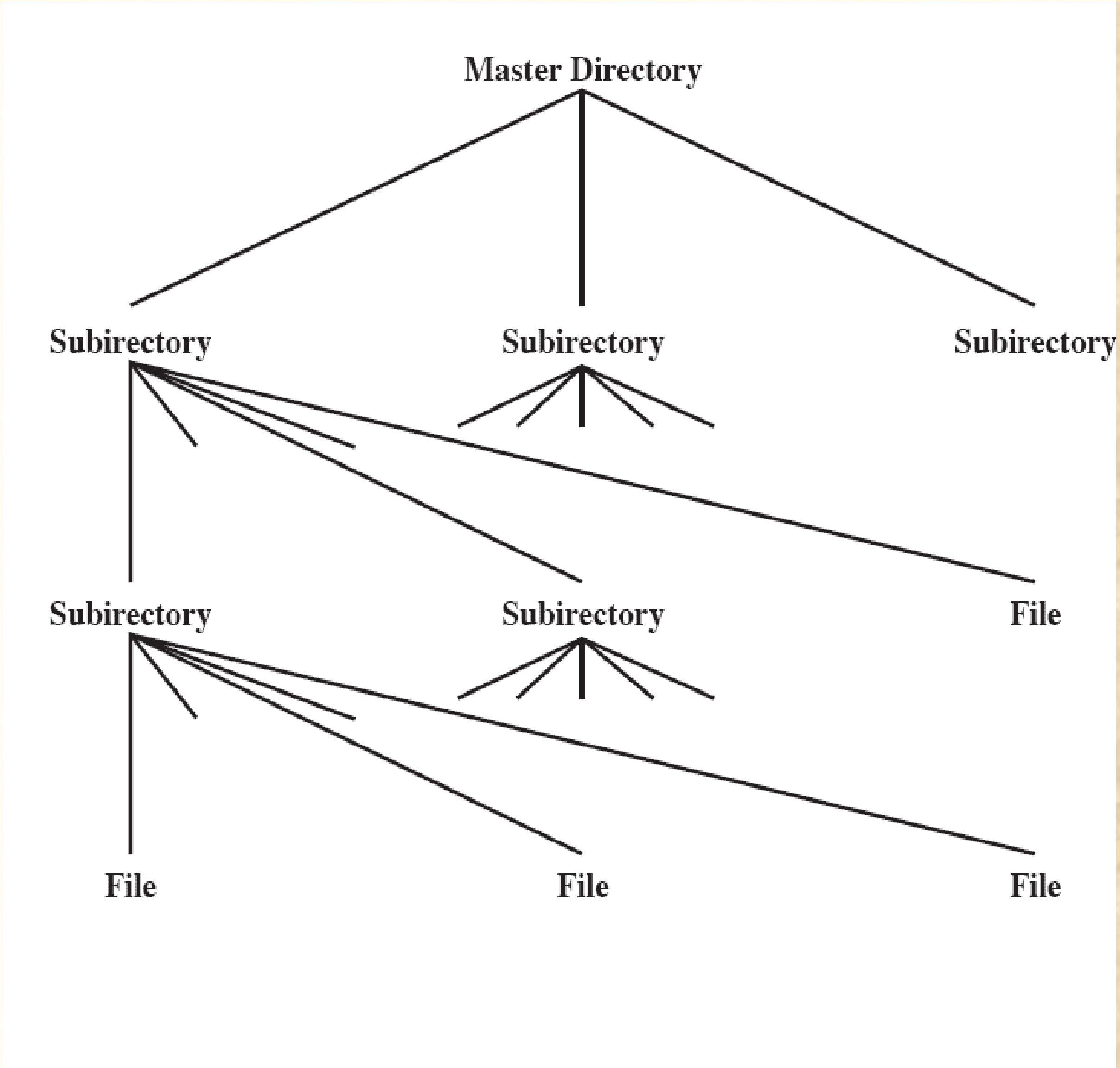
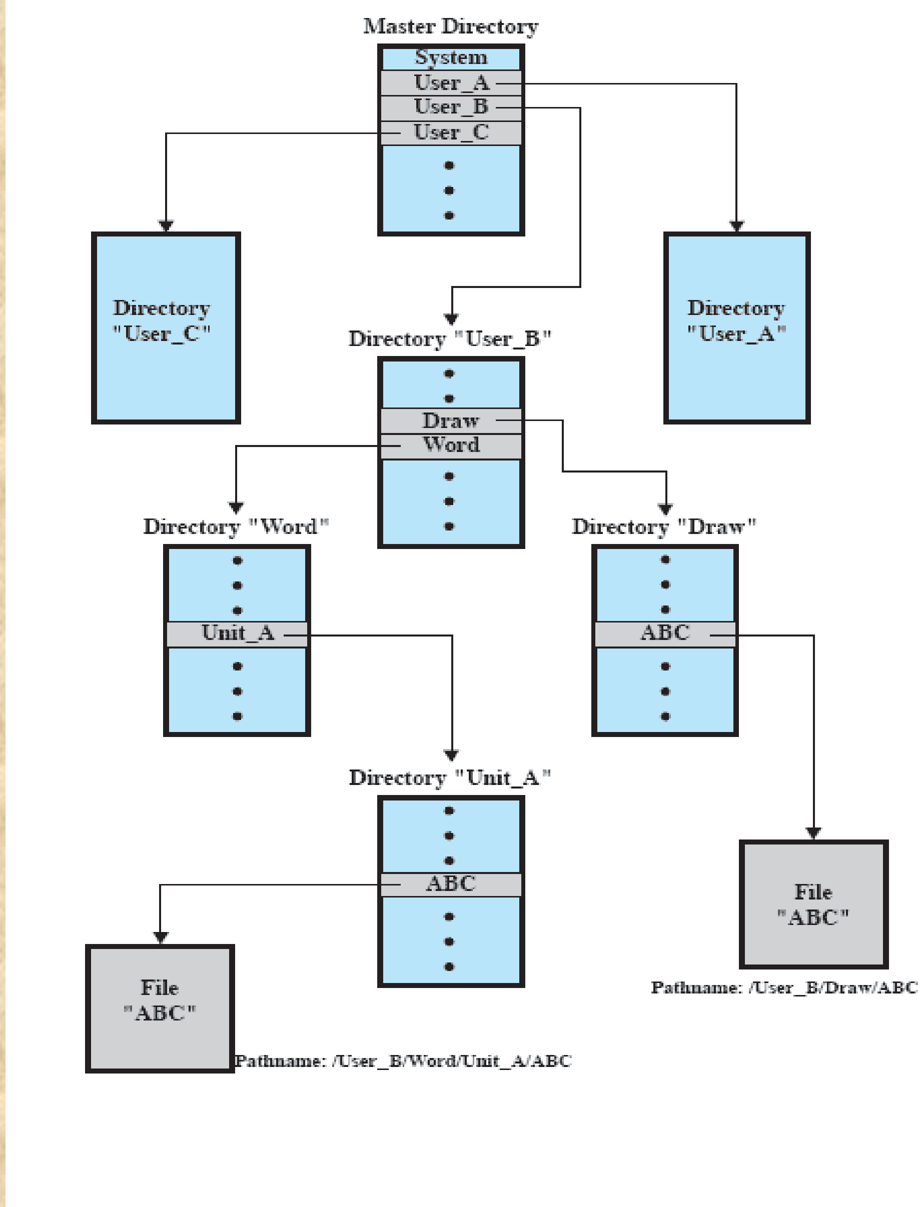


Figure 12.7 Example of Tree-Structured Directory



File Sharing

Two issues arise when allowing files to be shared among a number of users:

- access rights
- management of simultaneous access

Access Rights

- *None*
  - the user would not be allowed to read the user directory that includes the file
- *Knowledge*
  - the user can determine that the file exists and who its owner is and can then petition the owner for additional access rights
- *Execution*
  - the user can load and execute a program but cannot copy it
- *Reading*
  - the user can read the file for any purpose, including copying and execution
- *Appending*
  - the user can add data to the file but cannot modify or delete any of the file's contents
- *Updating*
  - the user can modify, delete, and add to the file's data
- *Changing protection*
  - the user can change the access rights granted to other users
- *Deletion*
  - the user can delete the file from the file system

User Access Rights

Owner

- usually the initial creator of the file
- has full rights
- may grant rights to others

Specific Users

- individual users who are designated by user ID

User Groups

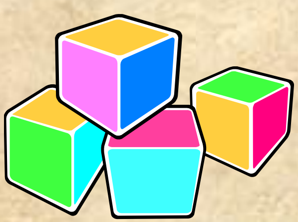
- a set of users who are not individually defined

All

- all users who have access to this system
- these are public files

Record Blocking

- Blocks are the unit of I/O with secondary storage
  - for I/O to be performed records must be organized as blocks



- Given the size of a block, three methods of blocking can be used:

1) **Fixed-Length Blocking** – fixed-length records are used, and an integral number of records (or bytes) are stored in a block
   Internal fragmentation – unused space at the end of each block for records, but not for bytes
   appropriate for byte-stream files.

2) **Variable-Length Spanned Blocking** – variable-length records are packed into blocks with no unused space

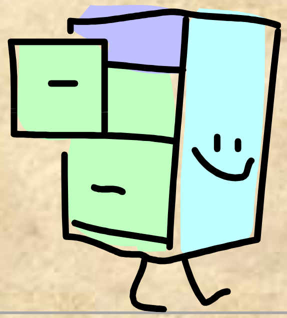
3) **Variable-Length Unspanned Blocking** – variable-length records are used, but spanning is not

File Allocation

- Disks are divided into *physical* blocks (sectors on a track)
- Files are divided into *logical* blocks (subdivisions of the file)
- Logical block size = some multiple of a physical block size
- The operating system or file management system is responsible for allocating blocks to files
- Space is allocated to a file as one or more ***portions*** (one or more contiguous disk blocks). A portion is the logical block size.
- ***File allocation table (FAT):***
  - A generic term for the data structure used to keep track of the disk portions assigned to a file

Preallocation vs Dynamic Allocation

- A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request
- For many applications it is difficult to estimate reliably the maximum potential size of the file
  - tends to be wasteful because users and application programmers tend to overestimate size
- Dynamic allocation allocates space to a file in portions as needed



Portion Size

- In choosing a portion size there is a trade-off between efficiency from the point of view of a single file versus overall system efficiency
- Items to be considered:
  1) contiguity of space increases performance, especially for `Retrieve_Next` operations (sequential access).
  2) having a large number of small portions increases the size of tables needed to manage the allocation information
  3) having fixed-size portions simplifies the reallocation of space
  4) having variable-size or small fixed-size portions minimizes waste of unused storage due to overallocation

Summarizing the Alternatives

- Two major alternatives:

Variable, large contiguous portions

- provides better performance, esp. for sequential access
- the variable size avoids waste
- the file allocation tables are small

Blocks

- small fixed portions provide greater flexibility
- they may require large tables or complex structures for their allocation
- contiguity has been abandoned as a primary goal
- blocks are allocated as needed

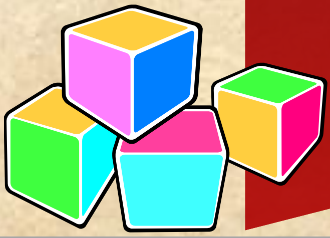
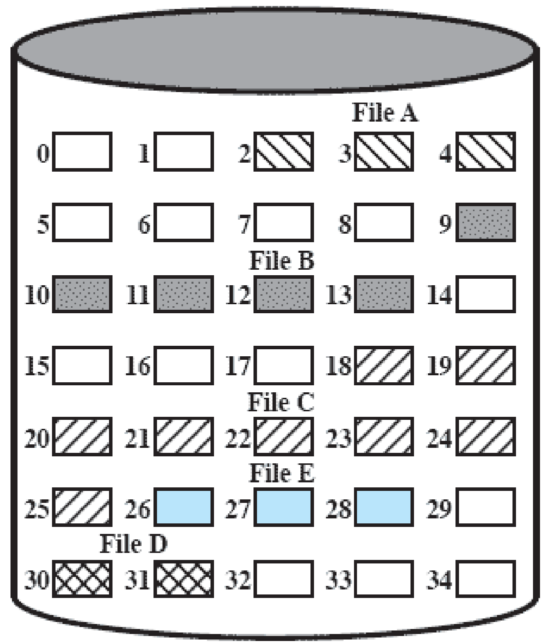


Table 12.3
File Allocation Methods

	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

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
- Is the best from the point of view of the individual sequential file

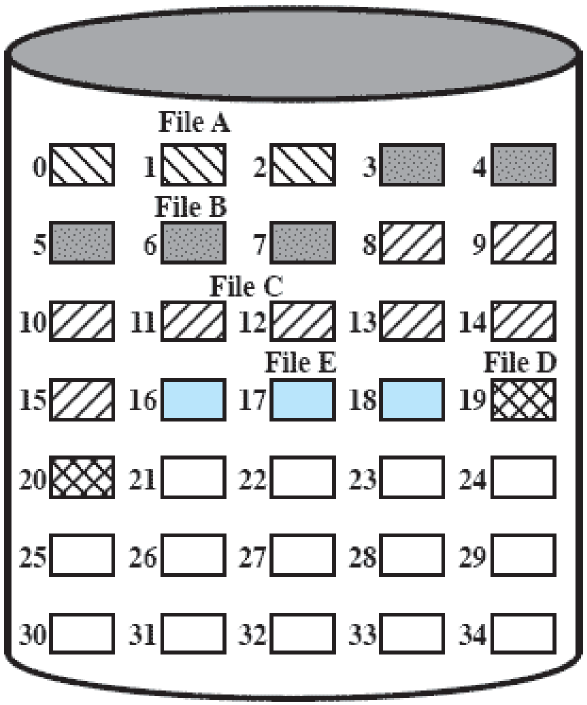


File Allocation Table

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 12.9 Contiguous File Allocation

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 12.10 Contiguous File Allocation (After Compaction)

Chained Allocation

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- The file allocation table needs just a single entry for each file
- No external fragmentation to worry about
- Better for sequential files

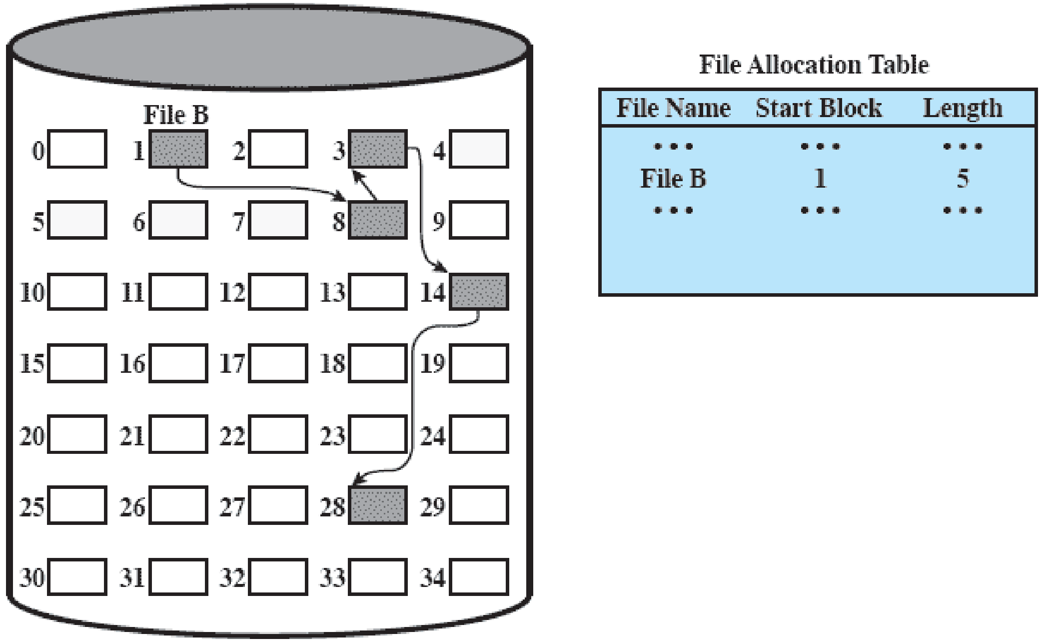
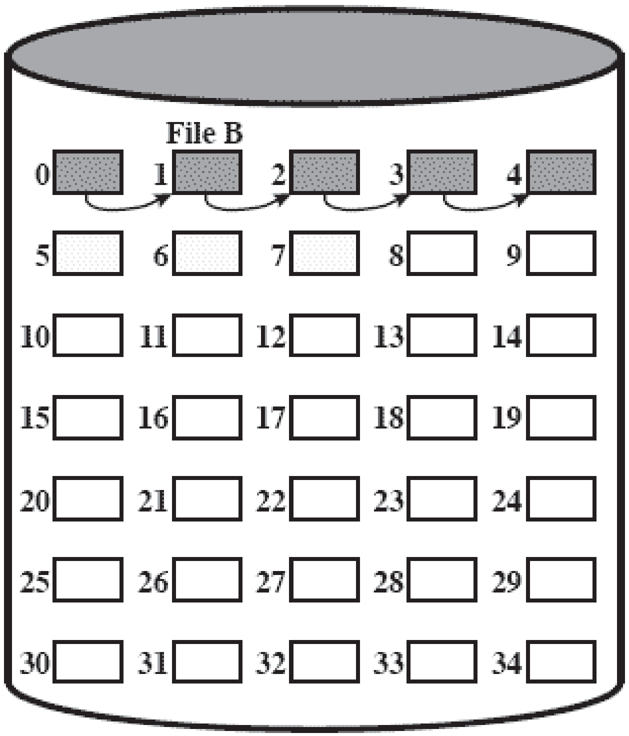


Figure 12.11 **Chained Allocation**

Chained Allocation After Consolidation

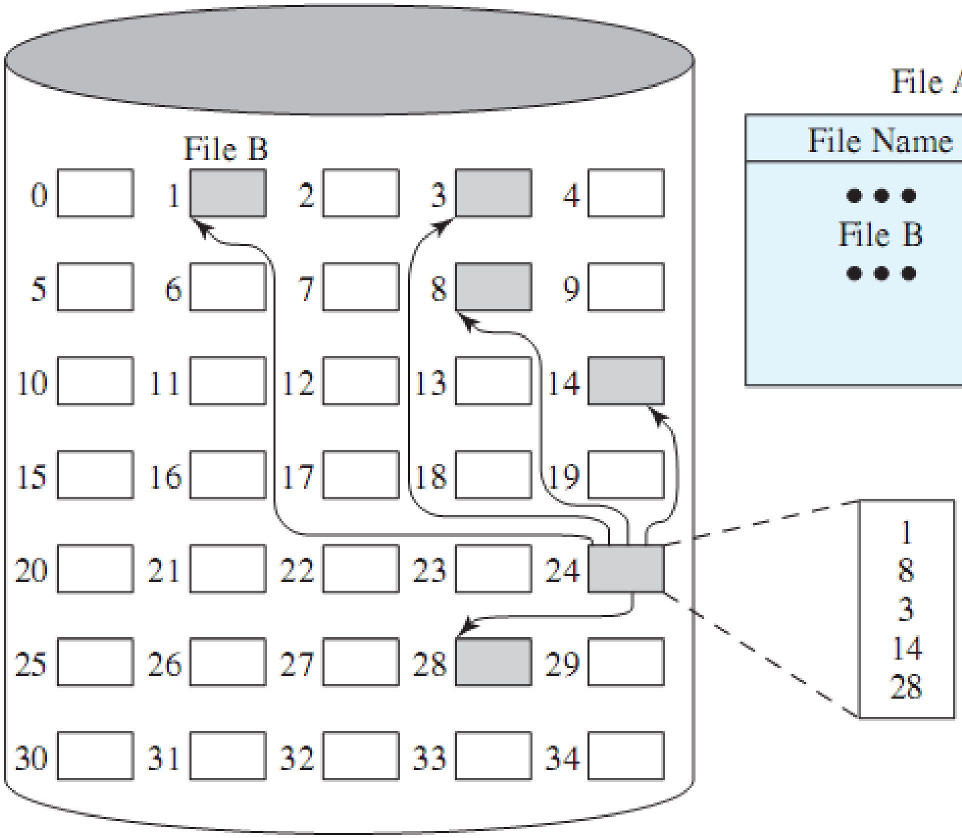


File Allocation Table

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

Figure 12.12 **Chained Allocation (After Consolidation)**

Indexed Allocation with Block Portions



File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

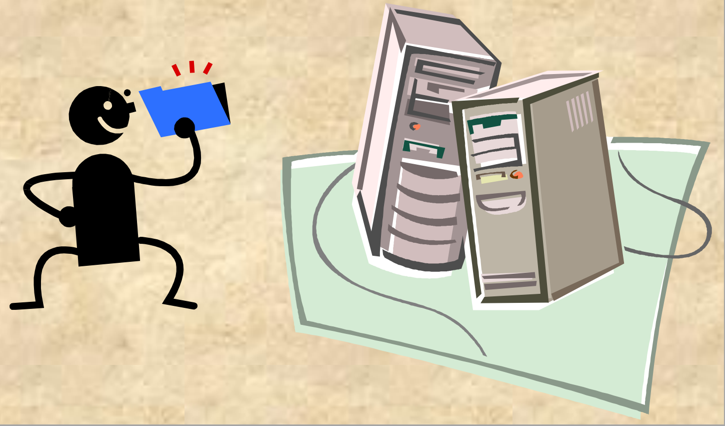
Figure 12.13 **Indexed Allocation with Block Portions**

Review

- File systems can support files organized as a sequence of bytes or as a sequence of records
- Access methods depend on file organization
- Disk storage of files can be contiguous, linked or indexed
- Logical blocks of a file are mapped to one or more disk sectors to create physical blocks (portions).
- Directories map user names to internal names
- File Allocation Tables map files to disk locations
- Free lists keep track of unallocated space.

Free Space Management

- Just as allocated space must be managed, so must the unallocated space
- 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
  - Bit vectors
  - Chained free portions
  - Indexing.
  - Free block list



Bit Tables (Bit Vectors)

- This method uses a vector containing one bit for each block on the disk
- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use

Advantages:

- works well with any file allocation method
- it is as small as possible

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

Disadvantages:

- leads to fragmentation
- every time you allocate a block you need to read the block first to recover the pointer to the new first free block before writing data to that block

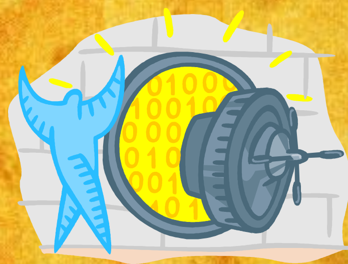
Indexing

- Treats free space as a file and uses an index table as it would for file allocation
- For efficiency, the free-space 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

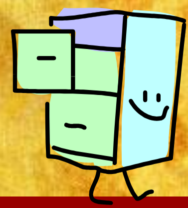
Free Block List

- Each block is assigned a number sequentially
  - the list of the numbers of all free blocks is maintained in a reserved portion of the disk
- Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number
  - the size of the free block list is 24 or 32 times the size of the corresponding bit table and must be stored on disk
- There are two effective techniques for storing a small part of the free block list in main memory:
  - the list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory
  - the list can be treated as a FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory

Volumes



- Essentially, 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

Summary

- **A file management system:**
  - is a set of system software that provides services to users and applications in the use of files
  - is typically viewed as a system service that is served by the operating system
- **Files:**
  - consist of a collection of records
  - if a file is primarily to be processed as a whole, a sequential file organization is the simplest and most appropriate
  - if sequential access is needed but random access to individual file is also desired, an indexed sequential file may give the best performance
  - if access to the file is principally at random, then an indexed file or hashed file may be the most appropriate
  - directory service allows files to be organized in a hierarchical fashion
- **Some sort of blocking strategy is needed**
- **Key function of file management scheme is the management of disk space**
  - strategy for allocating disk blocks to a file
  - maintaining a disk allocation table indicating which blocks are free