CSC 553 Operating Systems

Lecture 12 - File Management

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 proce**sses - 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

- Provide a means 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 Structure

- Four terms are commonly used when discussing files:
  – Field
  – Record
  – File
  – Database

Structure Terms

- File
  - Basic element of data
  - Contains a single value
  - Fixed or variable length
- Database
  - Collection of related data
  - Relationships among elements of data are explicit
  - Designed for use by a number of different applications
  - Consists of one or more types of files

Structure Terms

- File
  - Collection of similar records
  - Treated as a single entity
  - May be referenced by name
  - Access control restrictions usually apply at the file level
- Record
  - Collection of related fields that can be treated as a unit by some application program
  - Fixed or variable length

File Management System

- 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 Use 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 users' files
  4. Should be able to move data between files
  5. Should be able to back up and recover files in case of damage
  6. Should be able to access his or her files by name rather than by numeric identifier

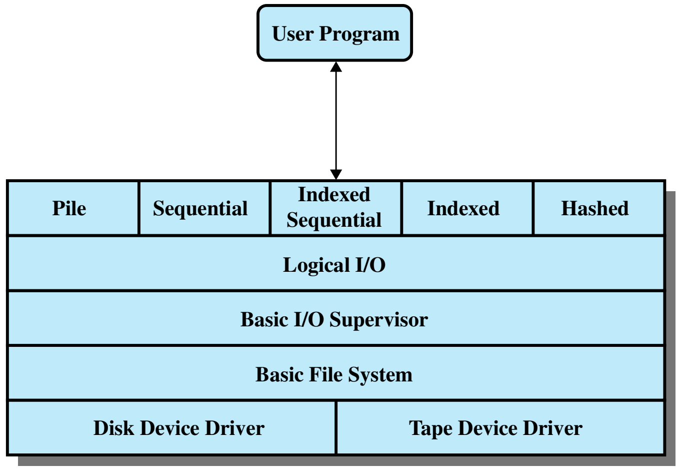


Figure 12.1 File System Software Architecture

Device Drivers

- Lowest level
- Communicates directly with peripheral devices
- Responsible for starting I/O operations on a device
- Processes the completion of an I/O request
- Considered to be part of the operating system

Basic File System

- Also referred to as the physical I/O level
- Primary interface with the environment outside the computer system
- Deals with blocks of data that are exchanged with disk or tape systems
- Concerned with the placement of blocks on the secondary storage device

Basic File System

- Concerned with buffering blocks in main memory
- Does not understand the content of the data or the structure of the files involved
- Considered part of the operating system

Basic I/O Supervisor

- Responsible for all file I/O initiation and termination
- At this level, control structures are maintained that deal with device I/O, scheduling, and file status
- Selects the device on which I/O is to be performed

Basic I/O Supervisor

- Concerned with scheduling disk and tape accesses to optimize performance
- I/O buffers are assigned and secondary memory is allocated at this level
- Part of the operating system

Logical I/O

- Enables users and applications to access records
- Provides general-purpose record I/O capability
- Maintains basic data about file

Access Method

- Level of the file system closest to the user
- Provides a standard interface between applications and the file systems and devices that hold the data
- Different access methods reflect different file structures and different ways of accessing and processing the data

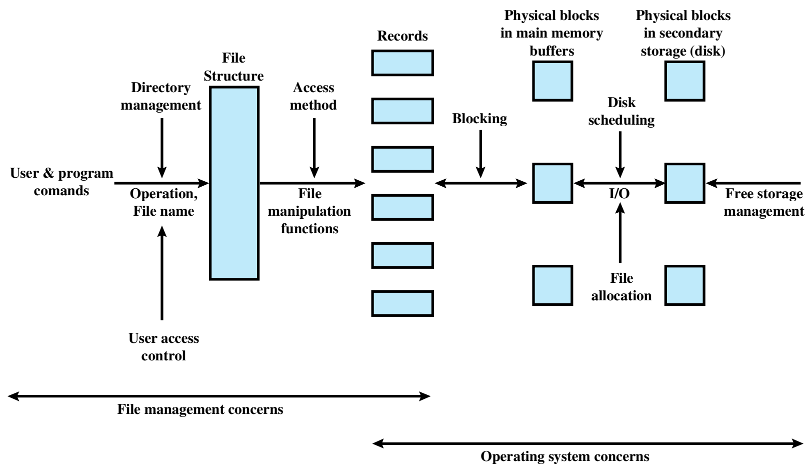


Figure 12.2 Elements of File Management

File Organization and Access

- ***File organization*** is the logical structuring of the records as determined by the way in which they are accessed

File Organization and Access

- 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 that will use the file

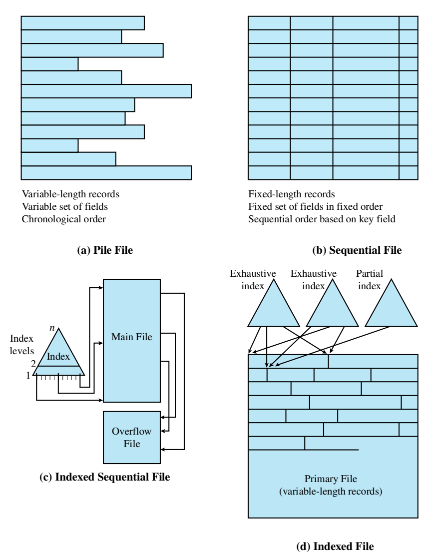


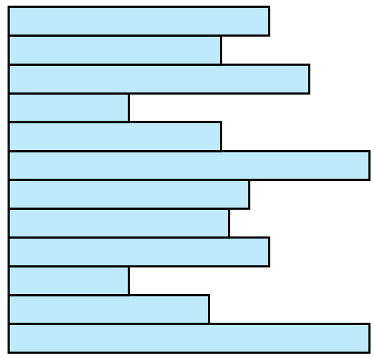
Figure 12.3 Common File Organizations

File Organization Types

- Five of the common file organizations are:
  - The pile
  - The sequential file
  - The indexed sequential file
  - The indexed file
  - The direct, or hashed, file

The Pile

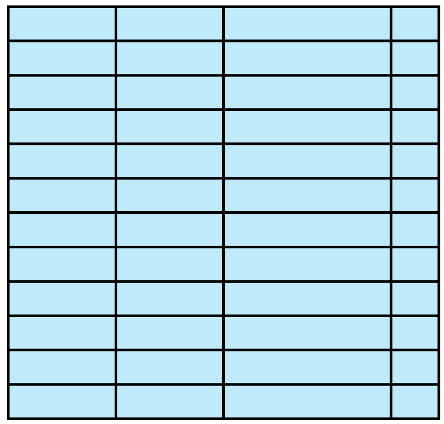
- Least complicated form of file organization
- Data are collected in the order they arrive
- Each record consists of one burst of data
- Purpose is simply to accumulate the mass of data and save it
- Record access is by exhaustive search



(a) Pile File

The Sequential File

- Most common form of file structure
- A fixed format is used for records
- Key field uniquely identifies the record
- Typically used in batch applications
- Only organization that is easily stored on tape as well as disk

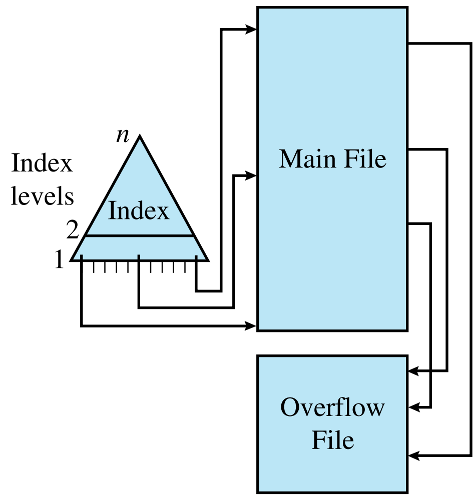


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

Indexed Sequential File

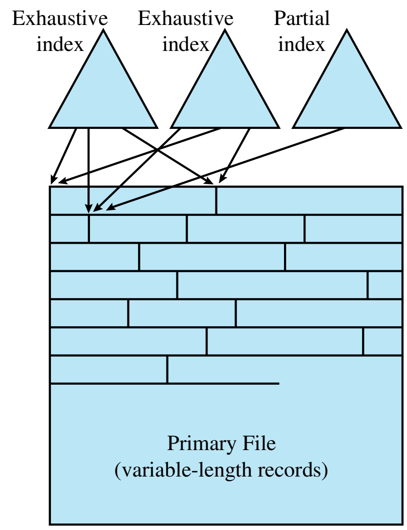
- Adds an index to the file to support random access
- Adds an overflow file
- Greatly reduces the time required to access a single record
- Multiple levels of indexing can be used to provide greater efficiency in access



(c) Indexed Sequential File

Indexed File

- Records are accessed only through their indexes
- 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

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

Direct or Hashed File

- Examples are:
  - Directories
  - Pricing tables
  - Schedules
  - Name lists

B-Trees

- A balanced tree structure with all branches of equal length
- Standard method of organizing indexes for databases
- Commonly used in OS file systems
- Provides for efficient searching, adding, and deleting of items

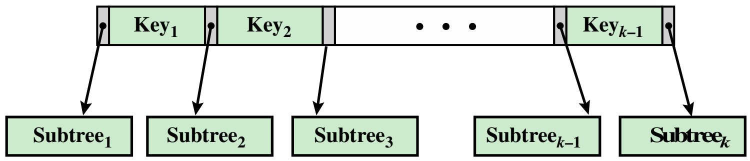


Figure 12.4 A B-tree Node with *k* Children

B-Tree Characteristics

- A B-tree is characterized by its minimum degree *d* and satisfies the following properties:
  - Every node has at most 2*d* – 1 keys and 2*d* children or, equivalently, 2*d* pointers
  - Every node, except for the root, has at least *d* – 1 keys and *d* pointers, as a result, each internal node, except the root, is at least half full and has at least *d* children

B-Tree Characteristics

- A B-tree is characterized by its minimum degree *d* and satisfies the following properties:
  - The root has at least 1 key and 2 children
  - All leaves appear on the same level and contain no information. This is a logical construct to terminate the tree; the actual implementation may differ
  - A nonleaf node with *k* pointers contains *k* – 1 keys

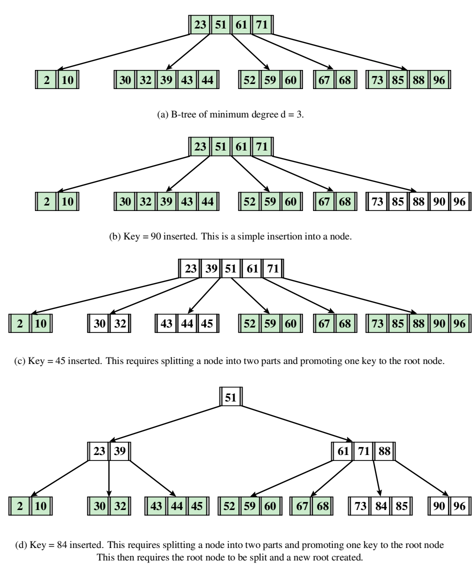


Figure 12.5 Inserting Nodes into a B-tree

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

Two-Level Scheme

- There is one directory for each user and a master directory
- Master directory has an entry for each user directory providing address and access control information
- 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

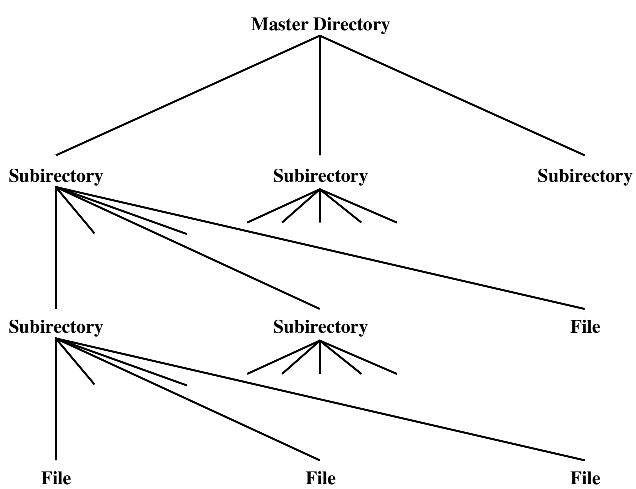


Figure 12.6 Tree-Structured Directory

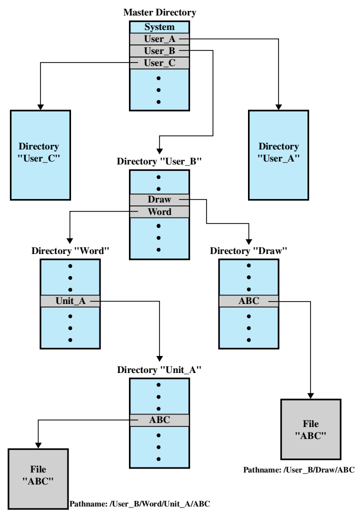


Figure 12.7 Example of Tree-Structured Directory

File Sharing

- Two issues arise when allowing files to be shared among a number of users:
  - Access rights
  - Management of simultaneous access

Access Rights

- **<u>None</u> -** The user would not be allowed to read the user directory that includes the file
- **<u>Knowledge</u> -** The user can determine that the file exists and who its owner is and can then petition the owner for additional access rights
- **<u>Execution</u> -** The user can load and execute a program but cannot copy it
- **<u>Reading</u> -** The user can read the file for any purpose, including copying and execution

Access Rights

- **<u>Appending</u> -** The user can add data to the file but cannot modify or delete any of the file's contents
- **<u>Updating</u> -** The user can modify, delete, and add to the file's data
- **<u>Changing protection - The user can change the access</u>** rights granted to other users
- **<u>Deletion</u> -** 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:
  - Fixed-Length Block
  - Variable-Length Spanned Bloc
  - Variable-Length Unspanned Blocking

Fixed-Length Block

- **Fixed-Length Blocking** – fixed-length records are used, and an integral number of records are stored in a block
- **Internal fragmentation** – unused space at the end of each block

Variable-Length Blocking

- **Variable-Length Spanned Blocking –** variable-length records are used and are packed into blocks with no unused space
- **Variable-Length Unspanned Blocking** – variable-length records are used, but spanning is not employed

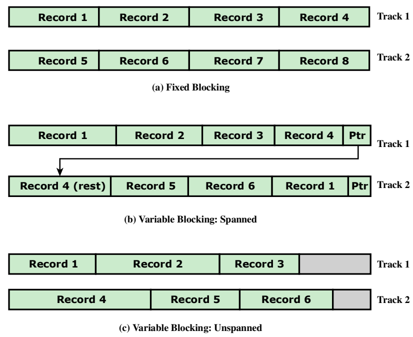


Figure 12.8 Record Blocking Methods

File Allocation

- On secondary storage, a file consists of a collection of blocks
- The operating system or file management system is responsible for allocating blocks to files
- The approach taken for file allocation may influence the approach taken for free space management

File Allocation

- Space is allocated to a file as one or more ***portions*** (contiguous set of allocated blocks)
- **File allocation table** ***(FAT)***
  - Data structure used to keep track of the 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

Portion Size

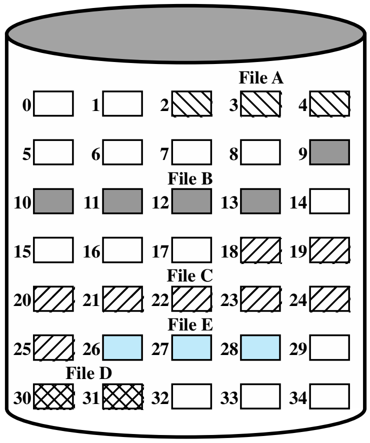
- Items to be considered:
  1) Contiguity of space increases performance, especially for Retrieve_Next operations, and greatly for transactions running in a transaction-oriented operating system
  2) Having a large number of small portions increases the size of tables needed to manage the allocation 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

Alternatives

- Two major alternatives:
  - Variable, large contiguous portions
    - Provides better performance
    - 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

File Allocation Methods

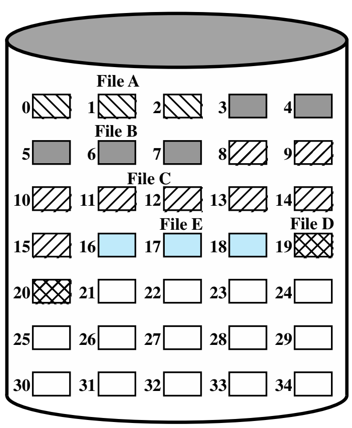
	Contiguous	**Chained**	**Indexed**	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table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



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

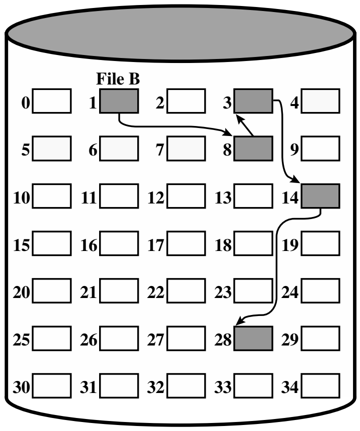
Figure 12.9 Contiguous File Allocation



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

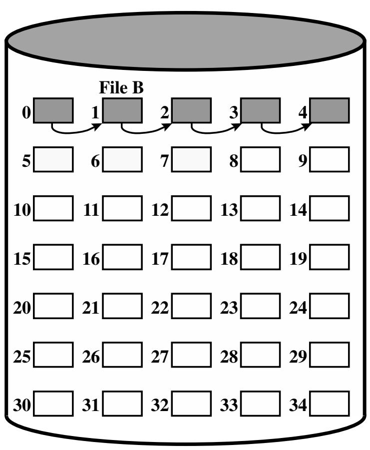
Figure 12.10 Contiguous File Allocation (After Compaction)



File Allocation Table

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

Figure 12.11 Chained Allocation



File Allocation Table

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

Figure 12.12 Chained Allocation (After Consolidation)

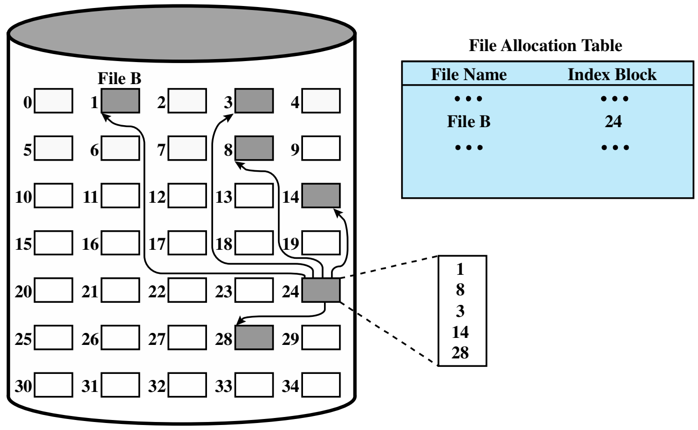


Figure 12.13 Indexed Allocation with Block Portions

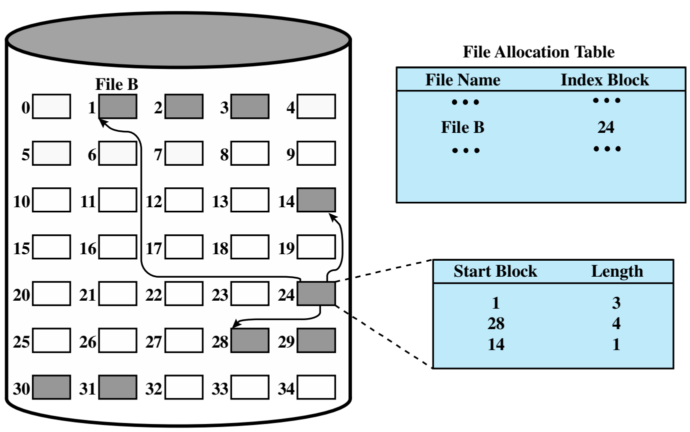


Figure 12.14 Indexed Allocation with Variable-Length Portions

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 Tables

- 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

Chained Free Portions

- 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 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

Free Block List

- 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

- 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

UNIX File Management

- In the UNIX file system, six types of files are distinguished:
  - **Regular, or ordinary -** Contains arbitrary data in zero or more data blocks
  - **Directory -** Contains a list of file names plus pointers to associated inodes (index nodes)
  - **Special** - Contains no data but provides a mechanism to map physical devices to file names

UNIX File Management

- In the UNIX file system, six types of files are distinguished:
  - **Named pipes -** An interprocess communications facility
  - **Links -** An alternative file name for an existing file
  - **Symbolic links -** A data file that contains the name of the file to which it is linked

Inodes

- All types of UNIX files are administered by the OS by means of inodes
- An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file
- Several file names may be associated with a single inode
  - An active inode is associated with exactly one file
  - Each file is controlled by exactly one inode

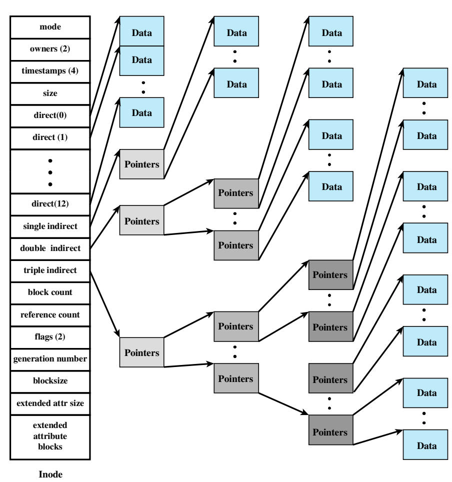


Figure 12.15 Structure of FreeBSD inode and File

File Allocation

- File allocation is done on a block basis
- Allocation is dynamic, as needed, rather than using preallocation
- An indexed method is used to keep track of each file, with part of the index stored in the inode for the file
- In all UNIX implementations the inode includes a number of direct pointers and three indirect pointers (single, double, triple)

Capacity of a FreeBSD File with 4-Kbyte Block Size

Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	512	2M
Double Indirect	$512 \times 512 = 256K$	1G
Triple Indirect	$512 \times 256K = 128M$	512G

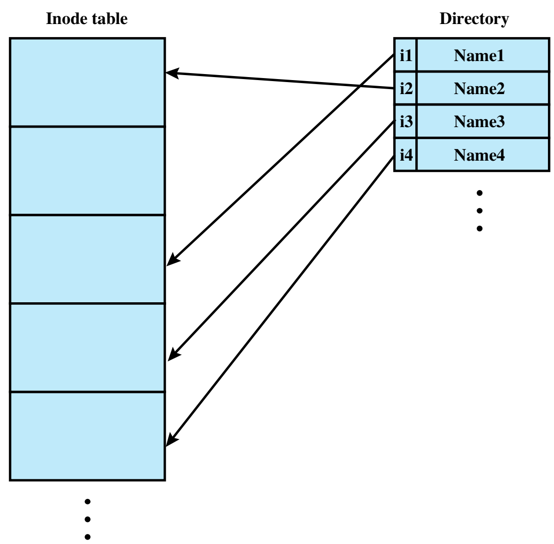


Figure 12.16 UNIX Directories and Inodes

Volume Structure

- A UNIX file system resides on a single logical disk or disk partition and is laid out with the following elements:
  - **Boot block** - Contains code required to boot the operating system
  - **Superblock** - Contains attributes and information about the file system
  - **Inode table** - Collection of inodes for each file
  - **Data blocks** - Storage space available for data files and subdirectories