UTN FRD – Sistemas Operativos
Unidad VII Sistemas de Archivos

Files

- Files are the central element to most applications
- The File System is one of the most important part of the OS to a user
- Desirable properties of files:
  - Long-term existence
  - Sharable between processes
  - Structure

File Management

- File management system consists of system utility programs that run as privileged applications
- Concerned with secondary storage

Typical Operations

- File systems also provide functions which can be performed on files, typically:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write

Terms

- Four terms are in common use when discussing files:
  - Field
  - Record
  - File
  - Database

Fields and Records

- Fields
  - Basic element of data
  - Contains a single value
  - Characterized by its length and data type
- Records
  - Collection of related fields
  - Treated as a unit

File and Database

- File
  - Have file names
  - Is a collection of similar records
  - Treated as a single entity
  - May implement access control mechanisms
- Database
  - Collection of related data
  - Relationships exist among elements
  - Consists of one or more files

File Management Systems

- Provides services to users and applications in the use of files
  - The way a user or application accesses files
- Programmer does not need to develop file management software

Objectives for a 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 lost or destroyed data
- Provide a standardized set of I/O interface routines to user processes
- Provide I/O support for multiple users (if needed)

Requirements for a general purpose system

1. Each user should be able to create, delete, read, write and modify files
2. Each user may have controlled access to other users' files
3. Each user may control what type of accesses are allowed to the users' files
4. Each user should be able to restructure the user's files in a form appropriate to the problem

Requirements cont.

5. Each user should be able to move data between files
6. Each user should be able to back up and recover the user’s files in case of damage
7. Each user should be able to access the user’s files by using symbolic names

Typical software organization

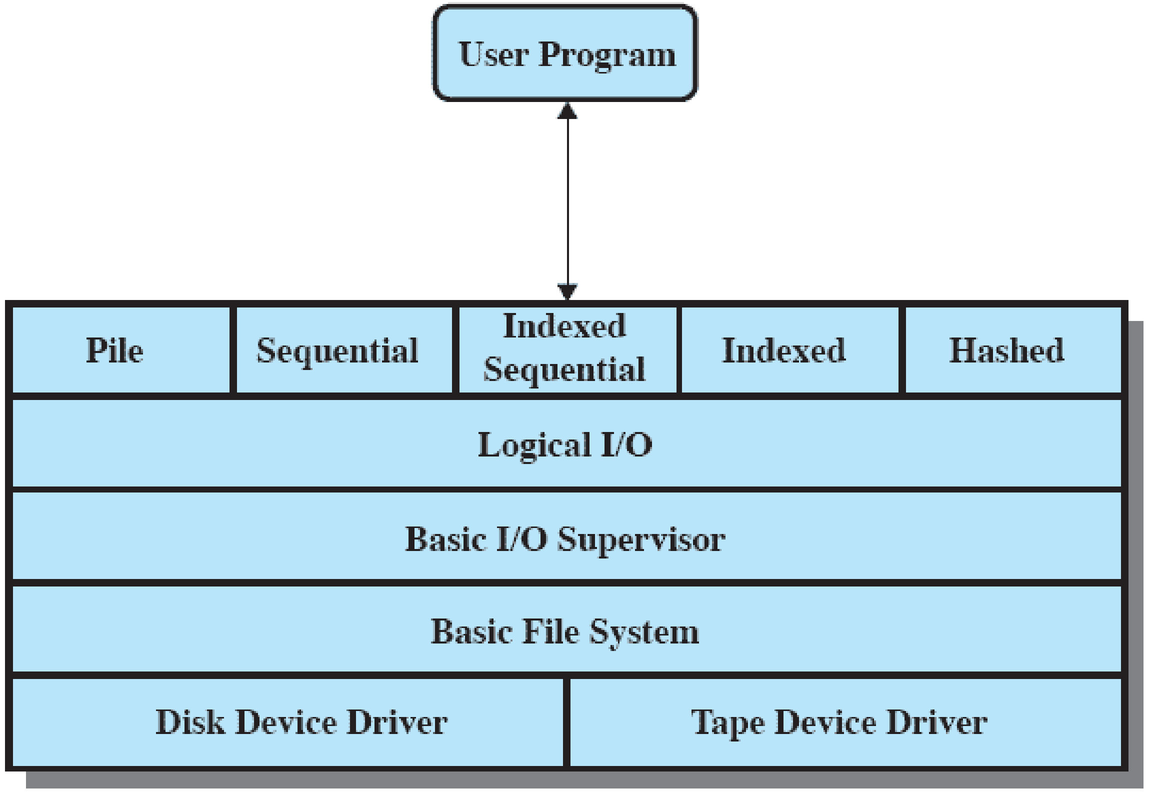


Figure 12.1 File System Software Architecture

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

Basic File System

- Physical I/O
- Primary interface with the environment outside the computer system
- Deals with exchanging blocks of data
- Concerned with the placement of blocks
- Concerned with buffering blocks in main memory

Basic I/O Supervisor

- Responsible for all file I/O initiation and termination.
- Control structures deal with
  - Device I/O,
  - Scheduling,
  - File status.
- Selects and schedules I/O with the device

Logical I/O

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

Access Method

- Closest to the user
- Reflect different file structures
- Provides a standard interface between applications and the file systems and devices that hold the data
- Access method varies depending on the ways to access and process data for the device.

Elements of File Management

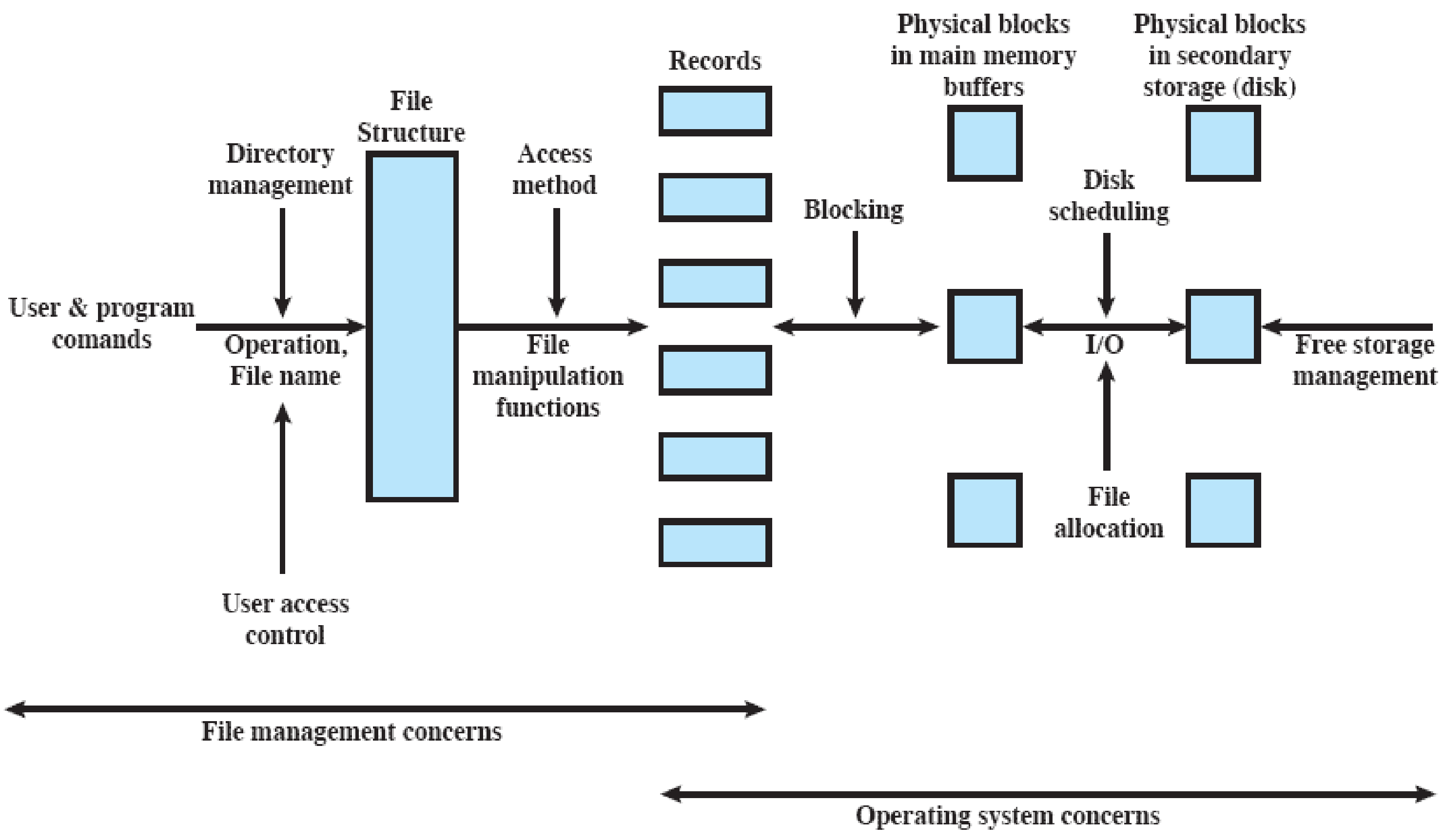


Figure 12.2 Elements of File Management

File Organization

- File Management Referring to the logical structure of records
  - Physical organization discussed later
- Determined by the ***way*** in which files are accessed

Criteria for File Organization

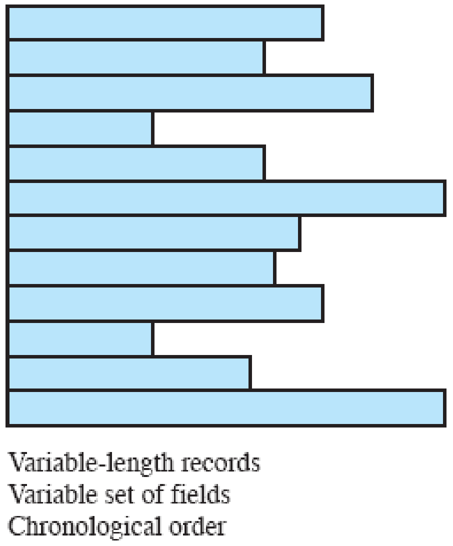
- Important criteria include:
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability
- Priority will differ depending on the use (e.g. read-only CD vs Hard Drive)
  - Some may even conflict

File Organisation Types

- Many exist, but usually variations of:
  - Pile
  - Sequential file
  - Indexed sequential file
  - Indexed file
  - Direct, or hashed, file

The Pile

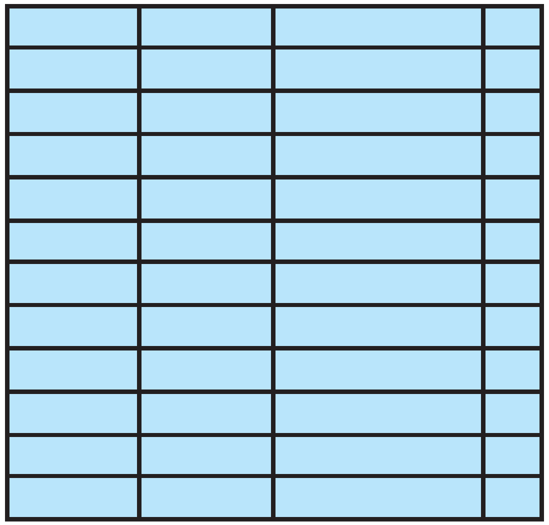
- Data are collected in the order they arrive
  - No structure
- Purpose is to accumulate a mass of data and save it
- Records may have different fields
- Record access is by exhaustive search



(a) Pile File

The Sequential File

- Fixed format used for records
- Records are the same length
- All fields the same (order and length)
- Field names and lengths are attributes of the file
- Key field
  - Uniquely identifies the record
  - Records are stored in key sequence



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

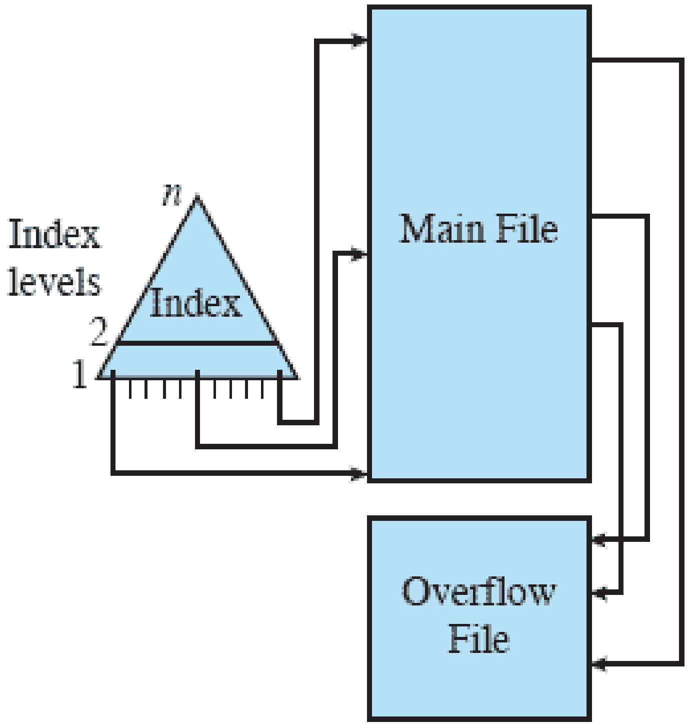
(b) Sequential File

Indexed Sequential File

- Maintains the key characteristic of the sequential file:
  - records are organized in sequence based on a key field.

Two features are added:

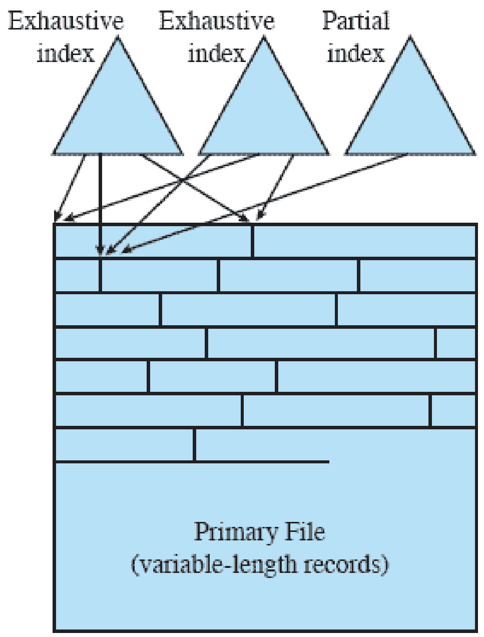
- an index to the file to suppor random access,
- and an overflow file.



(c) Indexed Sequential File

Indexed File

- Uses multiple indexes for different key fields
  - May contain an exhaustive index that contains one entry for every record in the main file
  - May contain a partial index
- When a new record is added to the main file, all of the index files must be updated.



(d) Indexed File

File Organization

- Access directly any block of a known address.
- The Direct or Hashed File
  - Directly access a block at a known address
  - Key field required for each record

Contents

- Contains information about files
  - Attributes
  - Location
  - Ownership
- Directory itself is a file owned by the operating system
- Provides mapping between file names and the files themselves

Directory Elements: Basic Information

- File Name
  - Name as chosen by creator (user or program).
  - Must be unique within a specific directory.
- File type
- File Organisation
  - For systems that support different organizations

Directory Elements: Address Information

- Volume
  - Indicates device on which file is stored
- Starting Address
- Size Used
  - Current size of the file in bytes, words, or blocks
- Size Allocated
  - The maximum size of the file

Directory Elements: Access Control Information

- Owner
  - The owner may be able to grant/deny access to other users and to change these privileges.
- Access Information
  - May include the user’s name and password for each authorized user.
- Permitted Actions
  - Controls reading, writing, executing, transmitting over a network

Directory Elements: Usage Information

- Date Created
- Identity of Creator
- Date Last Read Access
- Identity of Last Reader
- Date Last Modified
- Identity of Last Modifier
- Date of Last Backup
- Current Usage
  - Current activity, locks, etc

Simple Structure for a Directory

- The method for storing the previous information varies widely between systems
- Simplest is a list of entries, one for each file
  - Sequential file with the name of the file serving as the key
  - Provides no help in organizing the files
  - Forces user to be careful not to use the same name for two different files

Operations Performed on a Directory

- A directory system should support a number of operations including:
  - Search
  - Create files
  - Deleting files
  - Listing directory
  - Updating directory

Two-Level Scheme for a Directory

- One directory for each user and a master directory
  - Master directory contains entry for each user
  - Provides address and access control information
- Each user directory is a simple list of files for that user
  - Does not provide structure for collections of files

Hierarchical, or Tree-Structured Directory

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

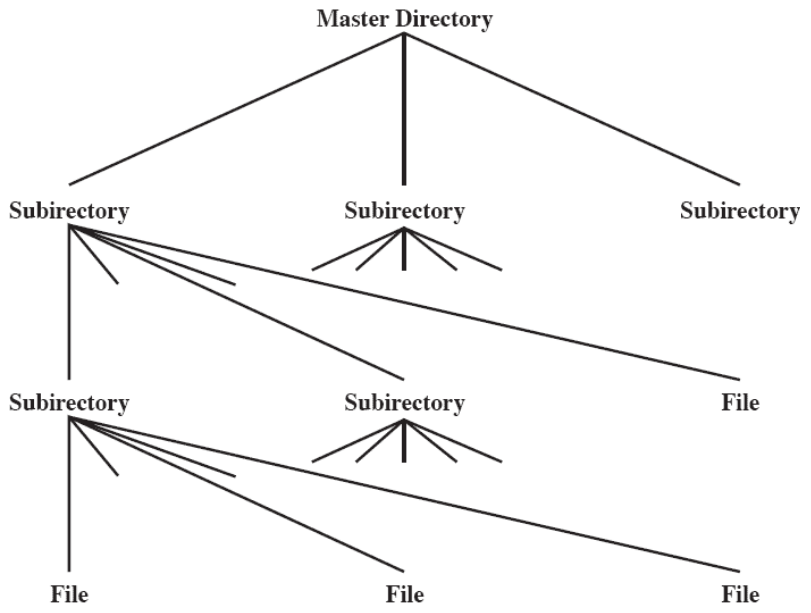


Figure 12.4 Tree-Structured Directory

Mostrar Directorio VMS

Naming

- Users need to be able to refer to a file by name
  - Files need to be named uniquely, but users may not be aware of all filenames on a system
- The tree structure allows users to find a file by following the directory path
  - Duplicate filenames are possible if they have different pathnames

Working Directory

- Stating the full pathname and filename is awkward and tedious
- Usually an interactive user or process is associated with a **current** or **working directory**
  - All file names are referenced as being relative to the working directory unless an explicit full pathname is used

File Sharing

- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights
  - Management of simultaneous access

Access Rights

- A wide variety of access rights have been used by various systems
  - often as a hierarchy where one right implies previous
- None
  - User may not even know of the files existence
- Knowledge
  - User can only determine that the file exists and who its owner is

Access Rights cont…

- 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

Access Rights cont…

- Updating
  - The user can modify, delete, and add to the file’s data.
- Changing protection
  - User can change access rights granted to other users
- Deletion
  - User can delete the file

User Classes

- Owner
  - Usually the files creator, usually has full rights
- Specific Users
  - Rights may be explicitly granted to specific users
- User Groups
  - A set of users identified as a group
- All
  - everyone

Simultaneous Access

- User may lock entire file when it is to be updated
- User may lock the individual records during the update
- Mutual exclusion and deadlock are issues for shared access

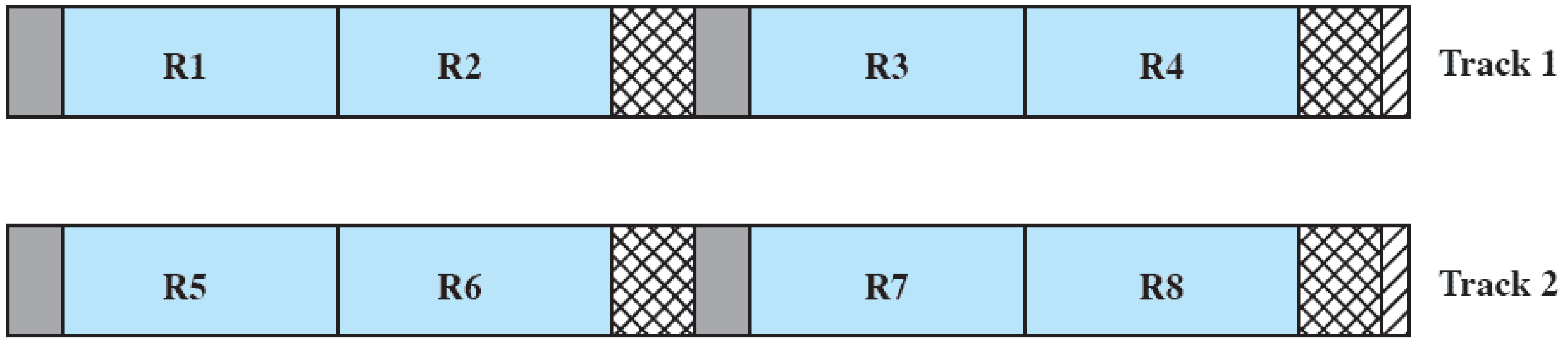
Blocks and records

- Records are the logical unit of access of a structured file
  - But blocks are the unit for I/O with secondary storage
- Three approaches are common
  - Fixed length blocking
  - Variable length spanned blocking
  - Variable-length unspanned blocking

Fixed Blocking

- Fixed-length records are used, and an integral number of records are stored in a block.
- Unused space at the end of a block is ***internal fragmentation***

Fixed Blocking



Fixed Blocking

🟦	Data	▧	Waste due to record fit to block size
◼	Gaps due to hardware design	▩	Waste due to block size constraint from fixed record size
▨	Waste due to block fit to track size		

Secondary Storage Management

- The Operating System is responsible for allocating blocks to files
- Two related issues
  - Space must be allocated to files
  - Must keep track of the space available for allocation

File allocation issues

1. When a file is created – is the maximum space allocated at once?
2. Space is added to a file in contiguous ‘portions’
   – What size should be the ‘portion’?
3. What data structure should be used to keep track of the file portions?

Preallocation vs Dynamic Allocation

- Need the maximum size for the file at the time of creation
- Difficult to reliably estimate the maximum potential size of the file
- Tend to overestimated file size so as not to run out of space

Portion size

- Two extremes:
  - Portion large enough to hold entire file is allocated
  - Allocate space one block at a time
- Trade-off between efficiency from the point of view of a single file, or the overall system efficiency

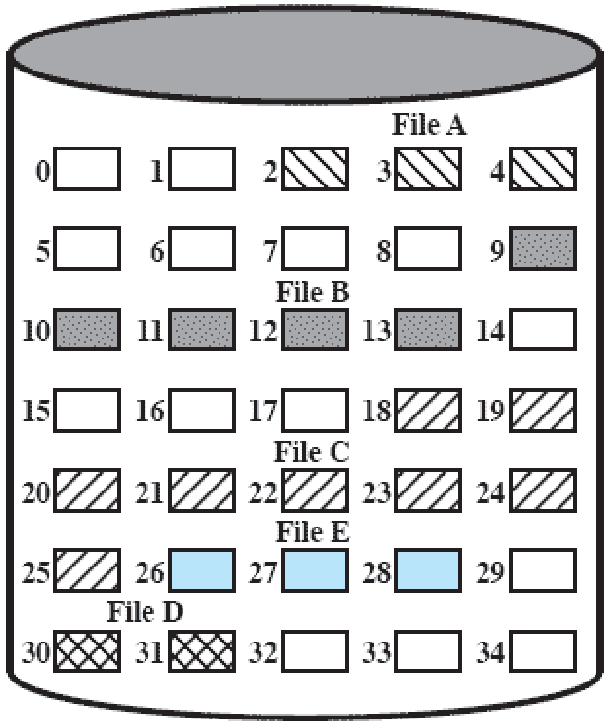
File Allocation Method

- Three methods are in common use:
  - contiguous,
  - chained, and
  - indexed.

Contiguous Allocation

- Single set of blocks is allocated to a file at the time of creation
- Only a single entry in the file allocation table
  - Starting block and length of the file
- External fragmentation will occur
  - Need to perform compaction

Contiguous File Allocation

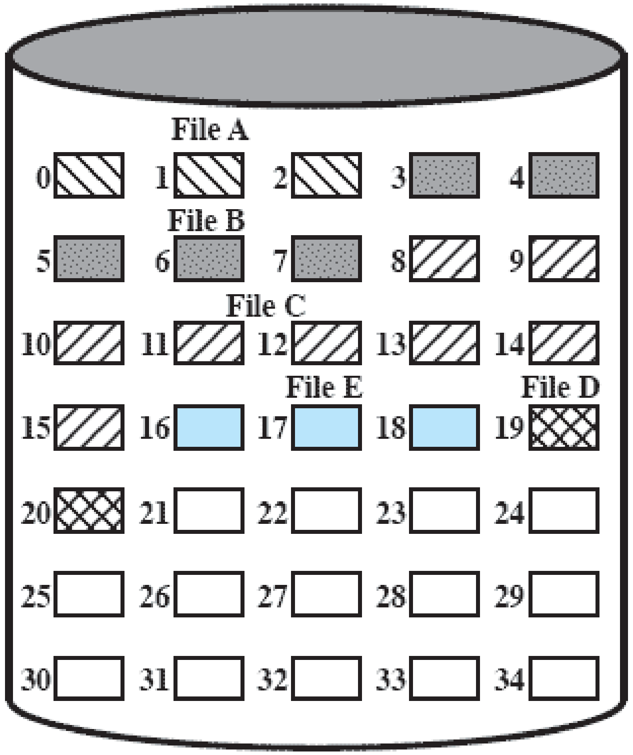


File Allocation Table

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

External fragmentation



File Allocation Table

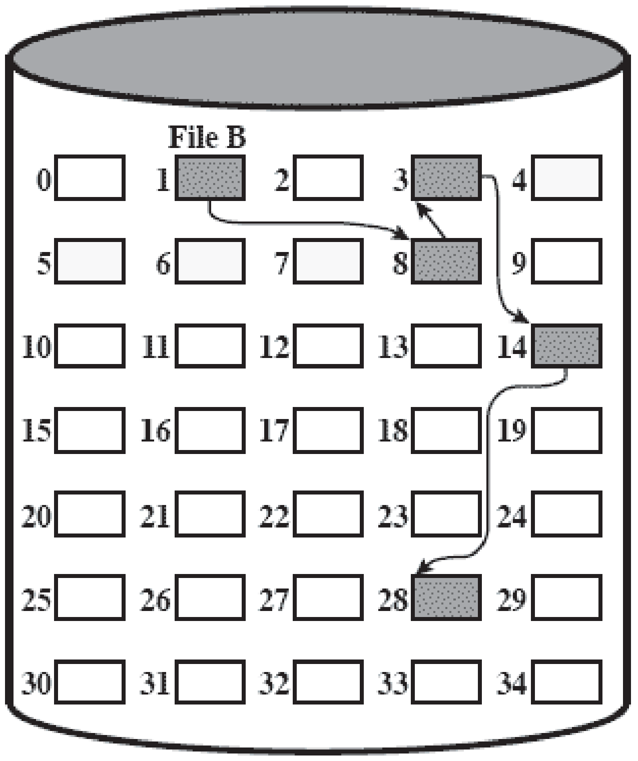
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

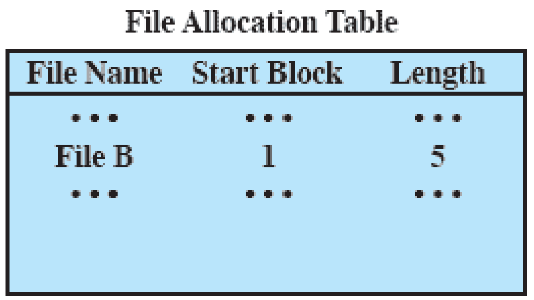
Figure 12.8 Contiguous File Allocation (After Compaction)

Chained Allocation

- Allocation on basis of individual block
- Each block contains a pointer to the next block in the chain
- Only single entry in the file allocation table
  - Starting block and length of file
- No external fragmentation
- Best for sequential files

Chained Allocation



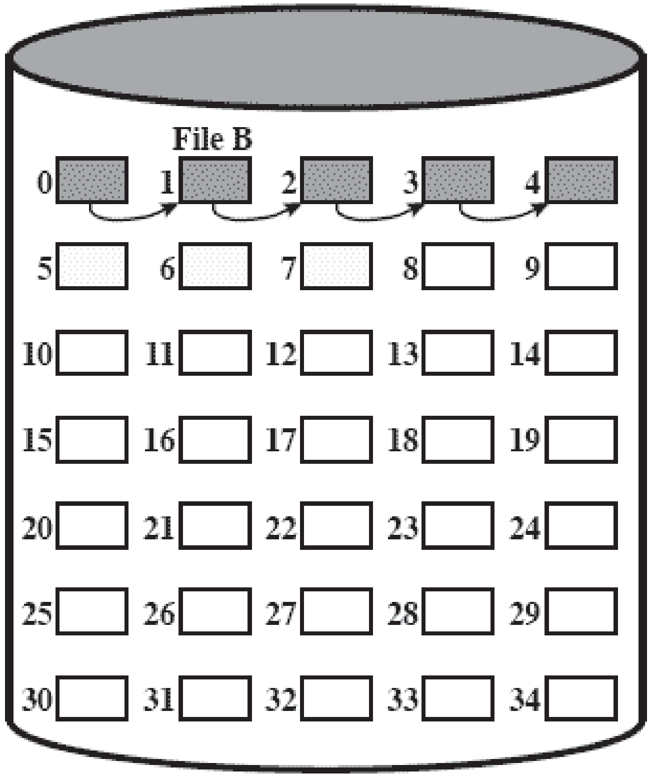


File Allocation Table

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

Figure 12.9 Chained Allocation

Chained Allocation Consolidation



File Allocation Table

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

Figure 12.10 Chained Allocation (After Consolidation)

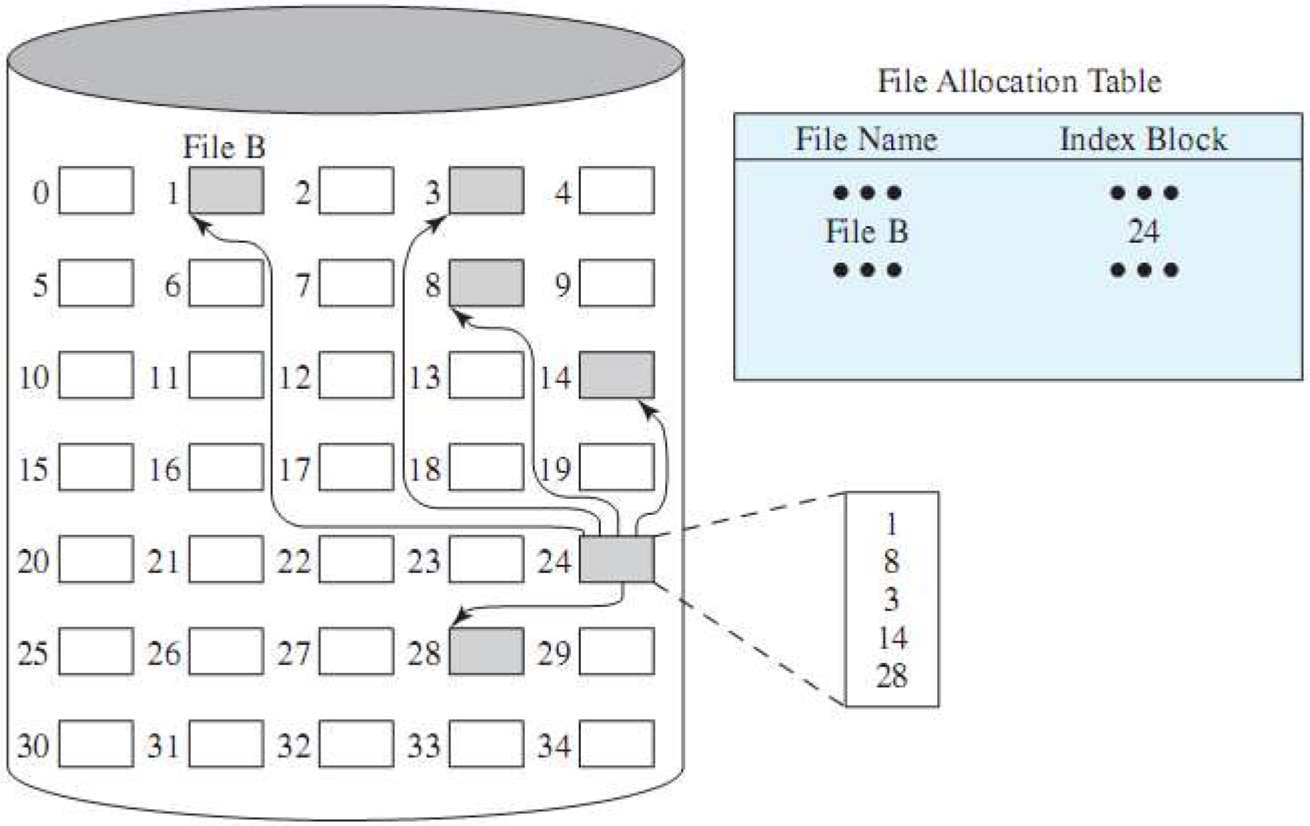
Indexed Allocation

- File allocation table contains a separate one-level index for each file
- The index has one entry for each portion allocated to the file
- The file allocation table contains block number for the index

Indexed Allocation Method

- Allocation may be either
  - Fixed size blocks or
  - Variable sized blocks
- Allocating by blocks eliminates external fragmentation
- Variable sized blocks improves locality
- Both cases require occasional consolidation

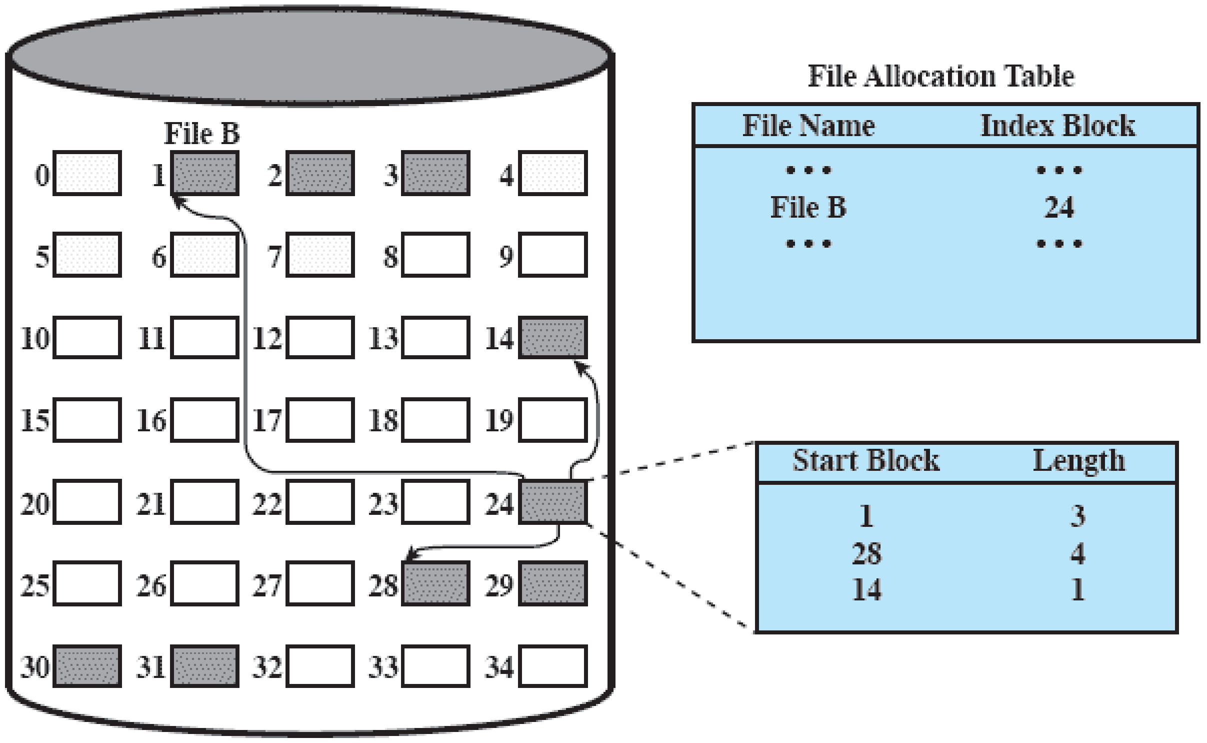
Indexed allocation with Block Portions



File Name	Index Block
● ● ●	● ● ●
File B	24
● ● ●	● ● ●

Figure 12.11 **Indexed Allocation with Block Portions**

Indexed Allocation with Variable Length Portions



File Allocation Table

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

Start Block	Length
1	3
28	4
14	1

Figure 12.12 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, we need to know which blocks are available.
- We need a disk allocation table 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
  - 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
- Suited to all file allocation methods
- Leads to fragmentation

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.
  - Thus, there is one entry in the table for every free portion on the disk.
- 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.

UNIX File Management

- Six types of files
  - Regular, or ordinary
  - Directory
  - Special
  - Named pipes
  - Links
  - Symbolic links

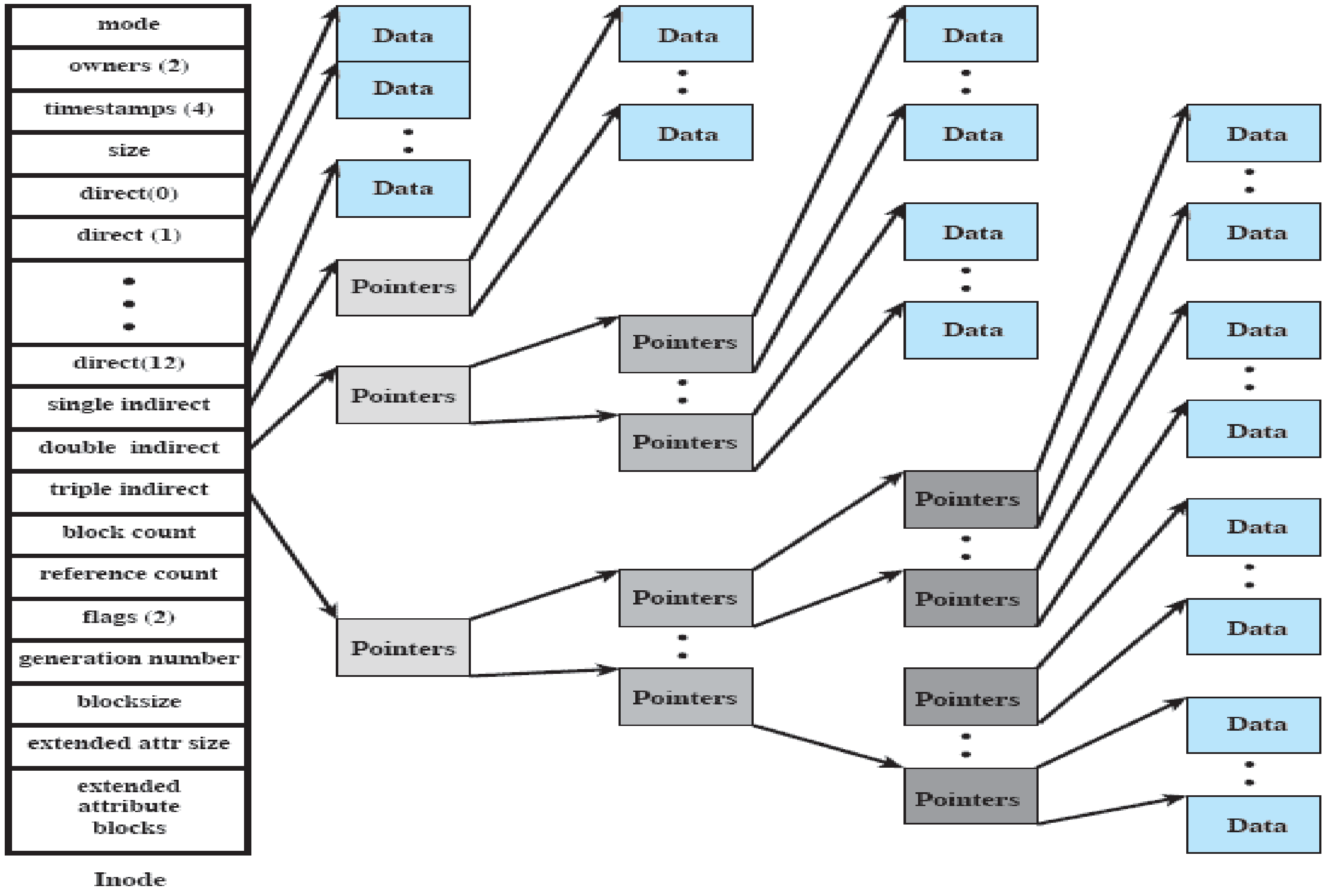
Inodes

- Index node
- Control structure that contains key information for a particular file.
- Several filenames may be associated with a single inode
  - But an active inode is associated with only one file, and
  - Each file is controlled by only one inode

Free BSD Inodes include:

- The type and access mode of the file
- The file’s owner and group-access identifiers
- Creation time, last read/write time
- File size
- Sequence of block pointers
- Number of blocks and Number of directory entries
- Blocksize of the data blocks
- Kernel and user setable flags
- Generation number for the file
- Size of Extended attribute information
- Zero or more extended attribute entries

FreeBSD Inode and File Structure



File Allocation

- File allocation is done on a block basis.
- Allocation is dynamic
  - Blocks may not be contiguous
- Index method keeps track of files
  - Part of index stored in the file inode.
- Inode includes a number of direct pointers
  - And three indirect pointers

UNIX Directories and Inodes

- Directories are files containing:
  - – a list of filenames
  - – Pointers to inodes

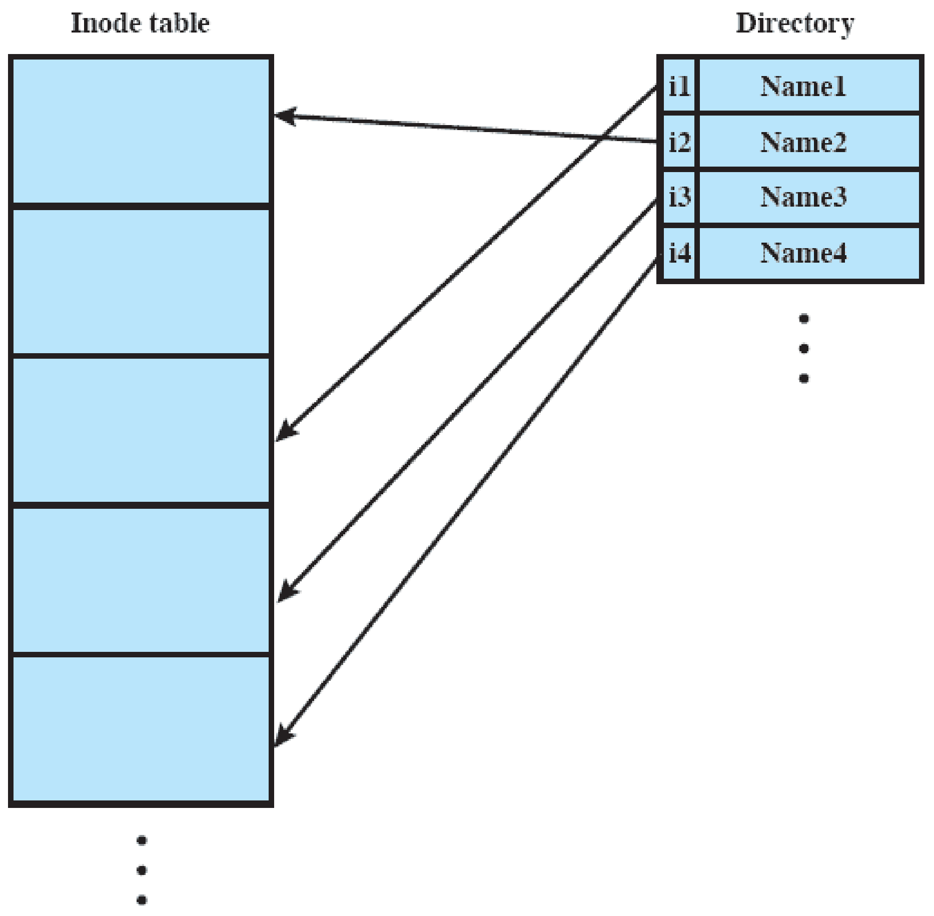


Figure 12.15 UNIX Directories and Inodes

An Example Program Using File System Calls (1/2)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                     /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);          /* ANSI prototype */

#define BUF_SIZE 4096                      /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                   /* protection bits for output file */

int main(int argc, char *argv[])
{
     int in_fd, out_fd, rd_count, wt_count;
     char buffer[BUF_SIZE];

     if (argc != 3) exit(1);               /* syntax error if argc is not 3 */
```

An Example Program Using File System Calls (2/2)

```
        /* Open the input file and create the output file */
        in_fd = open(argv[1], O_RDONLY);     /* open the source file */
        if (in_fd < 0) exit(2);               /* if it cannot be opened, exit */
        out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
        if (out_fd < 0) exit(3);              /* if it cannot be created, exit */

        /* Copy loop */
        while (TRUE) {
             rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;             /* if end of file or error, exit loop */
             wt_count = write(out_fd, buffer, rd_count); /* write data */
             if (wt_count <= 0) exit(4);      /* wt_count <= 0 is an error */
        }

        /* Close the files */
        close(in_fd);
        close(out_fd);
        if (rd_count == 0)                    /* no error on last read */
             exit(0);
        else
             exit(5);                         /* error on last read */
}
```