BCA
Fourth Semester
" Operating System"

Files

A file is named collection of related information normally resides on a secondary storage device such as disk or tape.

Commonly, files represent programs (both source and object forms) and data; data files may be numeric, alphanumeric, or binary.

Information stored in files must be persistent,- not be effected by power failures and system reboot.

The files are managed by OS.

The part of OS that is responsible to manage files is known as the file system.

File Attributes

A file is named, for the convenience of its human users, and it is referred to by its name. A name is string of characters.

The file attributes may very from system to system. Some common attributes are.

Name – only information kept in human-readable form
Identifier – unique tag (number) identifies file within file system
Type – needed for systems that support different types
Location – pointer to file location on device
Size – current file size
Protection – controls who can do reading, writing, executing
Time, date, and user identification – data for protection, security, and usage monitoring

Information about files are kept in the directory structure, which is maintained on the disk

File Operations

OS provides system calls to perform operations on files. Some common calls are:

Create: If disk space is available it create new file without data.

Delete: Deletes files to free up disk space.

Open: Before using a file, a process must open it.

Close: When all access are finished, the file should be closed to free up the internal table space.

Read: Reads data from file.

Append: Adds data at the end of the file.

Seek: Repositions the file pointer to a specific place in the file.

Get attributes: Returns file attributes for processing.

Set attributes: To set the user settable attributes when file changed.

Rename: Rename file.

File Types

Many OS supports several types of files

To operate on file OS need to identify the file type

Common technique to implement the file type is to include the file type as part of file name, i.e file name is split into two parts: name and extension, separated by period (.)

E.g: *example.c*

But Unix like OS use a magic number to identify file types.

Common file types

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

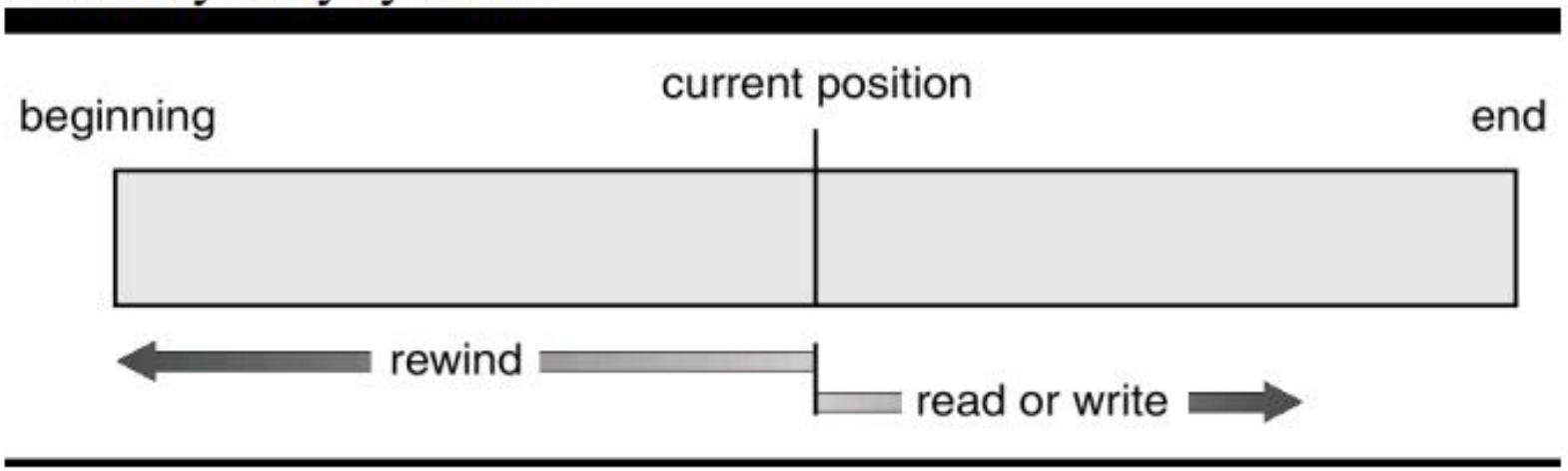
Sequential and Direct access

Sequential Access:

The simplest excess method; Information in the file is processed in order, one record after the other.

Convenient when the storage medium is magnetic tap.

Used in many early systems.



Sequential Access Operations:

read next - reads the next portion of the file and automatically advances a file pointer

write next - appends to the end of the file

reset – reset to the beginning

Access Methods

Direct Access:

Files whose bytes or records can be read in any order.

Based on disk model of file, since disks allow random access to any block.

Used for immediate access to large amounts of information.

When a query concerning a particular subject arrives, we compute which block contain the answer, and then read that block directly to provide desired information.

Operations:

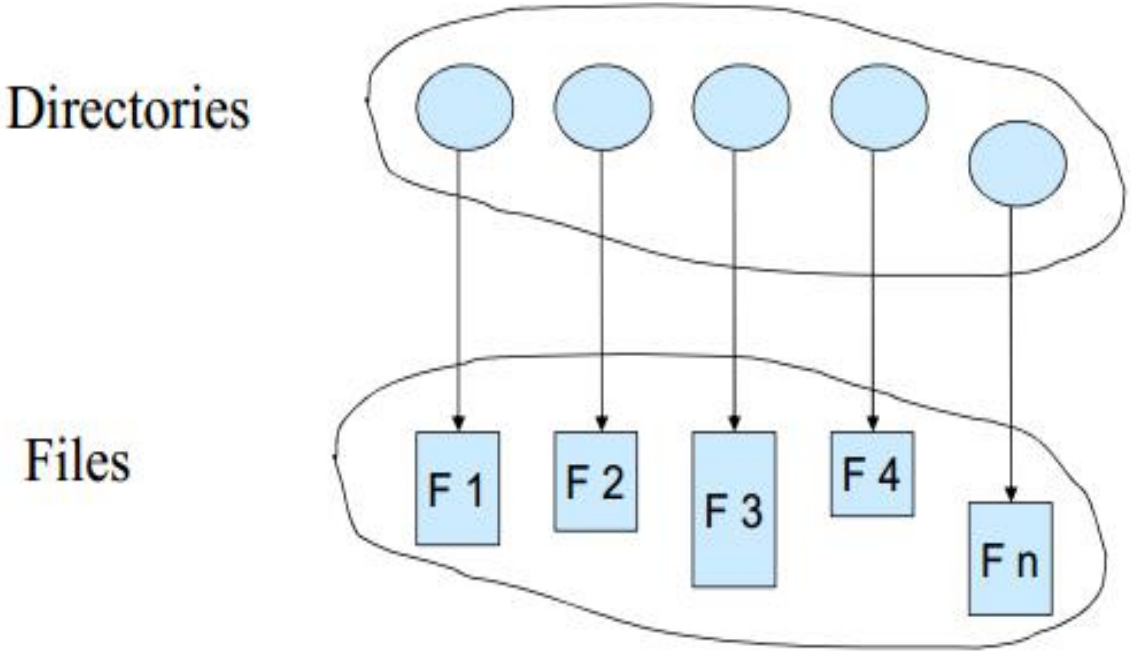
read *n*
write *n*
position to *n (seek)*
read next
write next
rewrite *n*

File can be access sequentially from the current position.

n = relative block number

Directory Structure

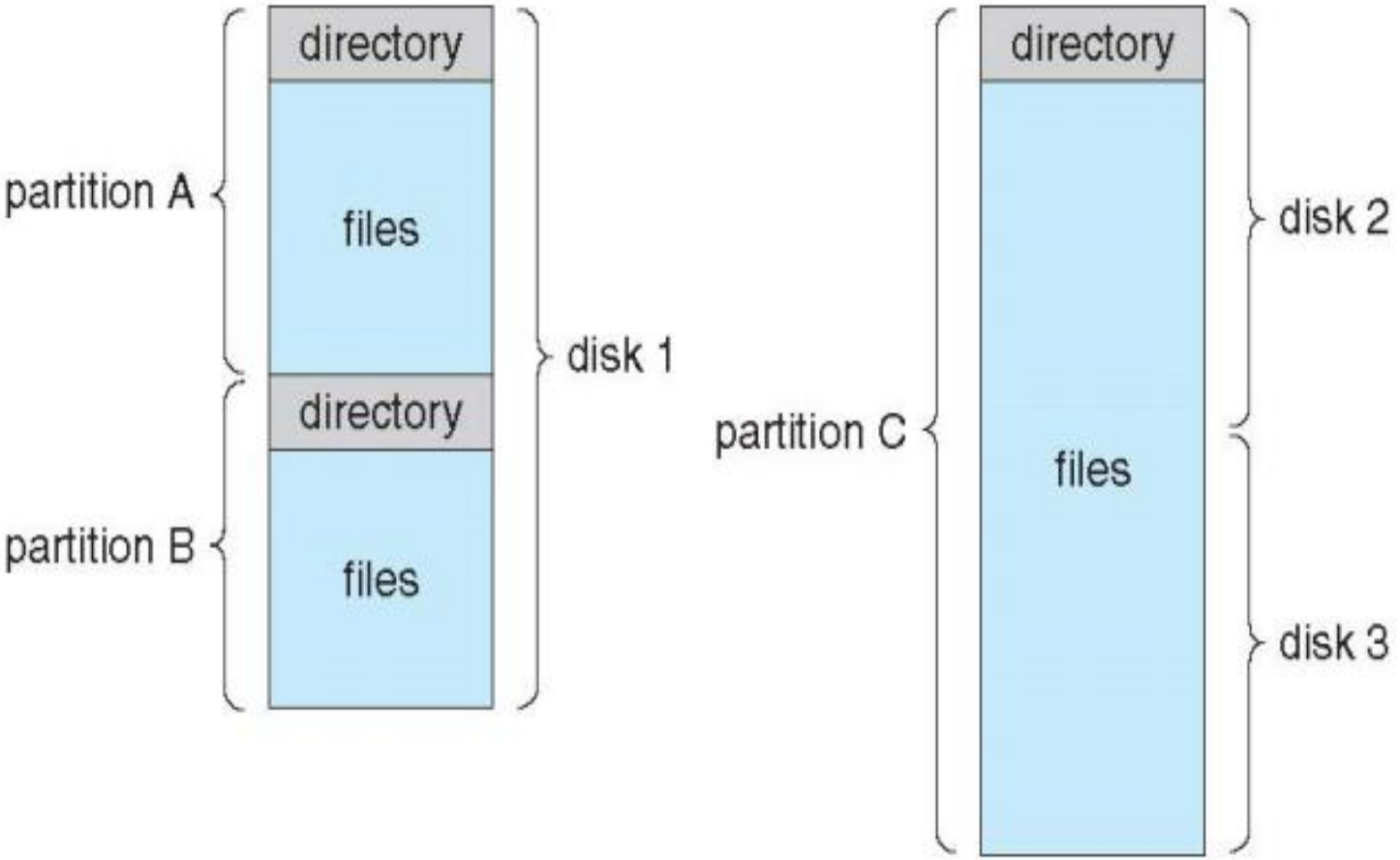
A directory is a node containing informations about files.



Both the directory structure and the files reside on disk
Directories can have different structures

Directory Structure

A Typical File-system Organization

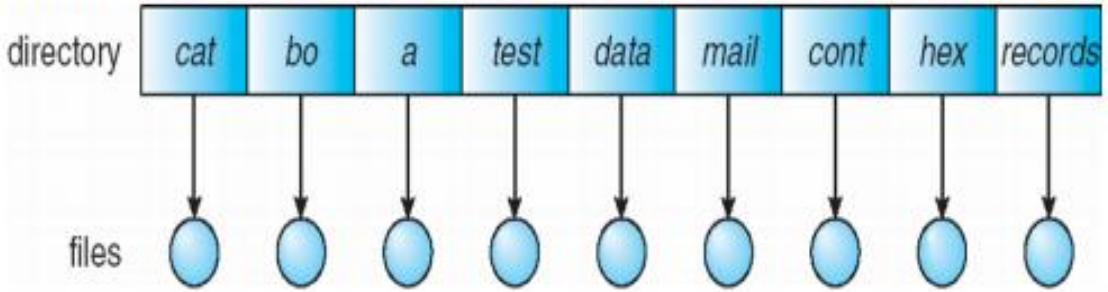


Directory Structure

Single-Level-Directory:

All files are contained in the same directory.

Easy to support and understand; but difficult to manage large amount of files and to manage different users.

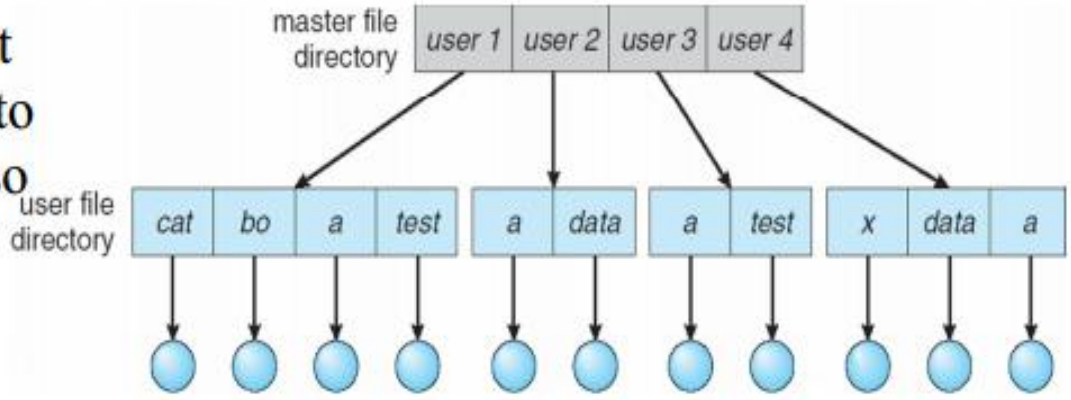


Two-Level-Directory:

Separate directory for each user.

Used on a multiuser computer and on a simple network computers.

It has problem when users want to cooperate on some task and to access one another's files. It also cause problem when a single user has large number of files.



Directory Structure

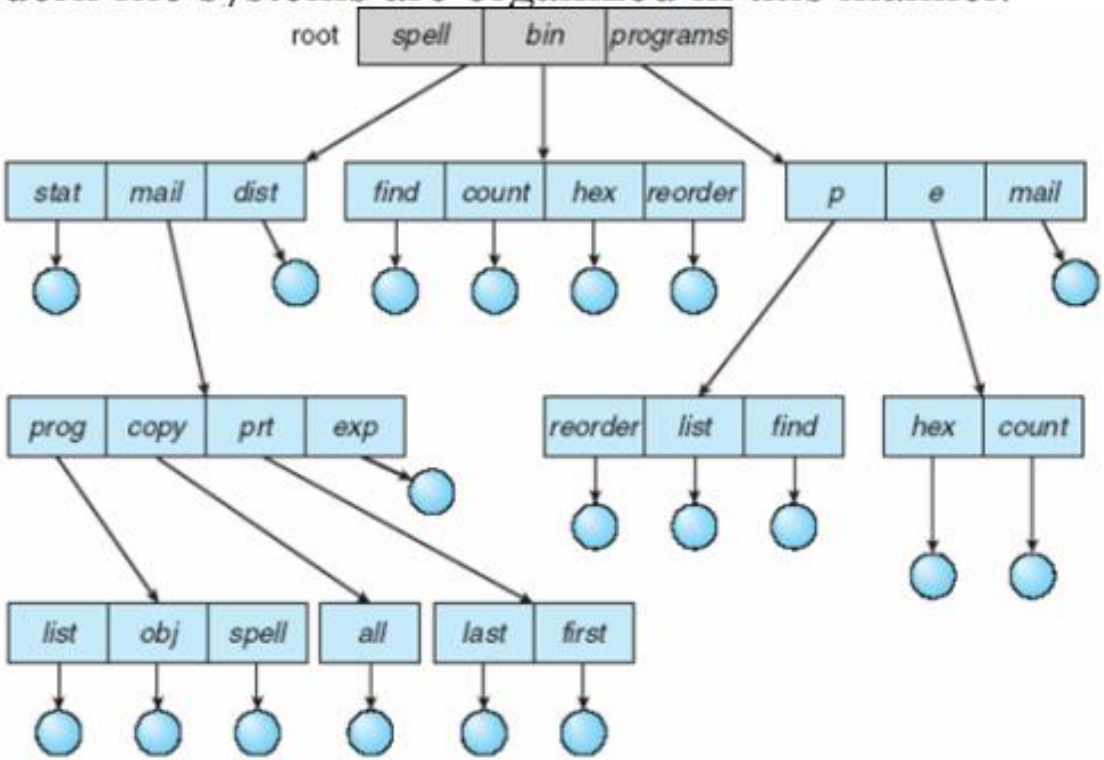
Tree-Structured Directories

Generalization of two-level-structure to a tree of arbitrary height.

This allow the user to create their own subdirectories and to organize their files accordingly.

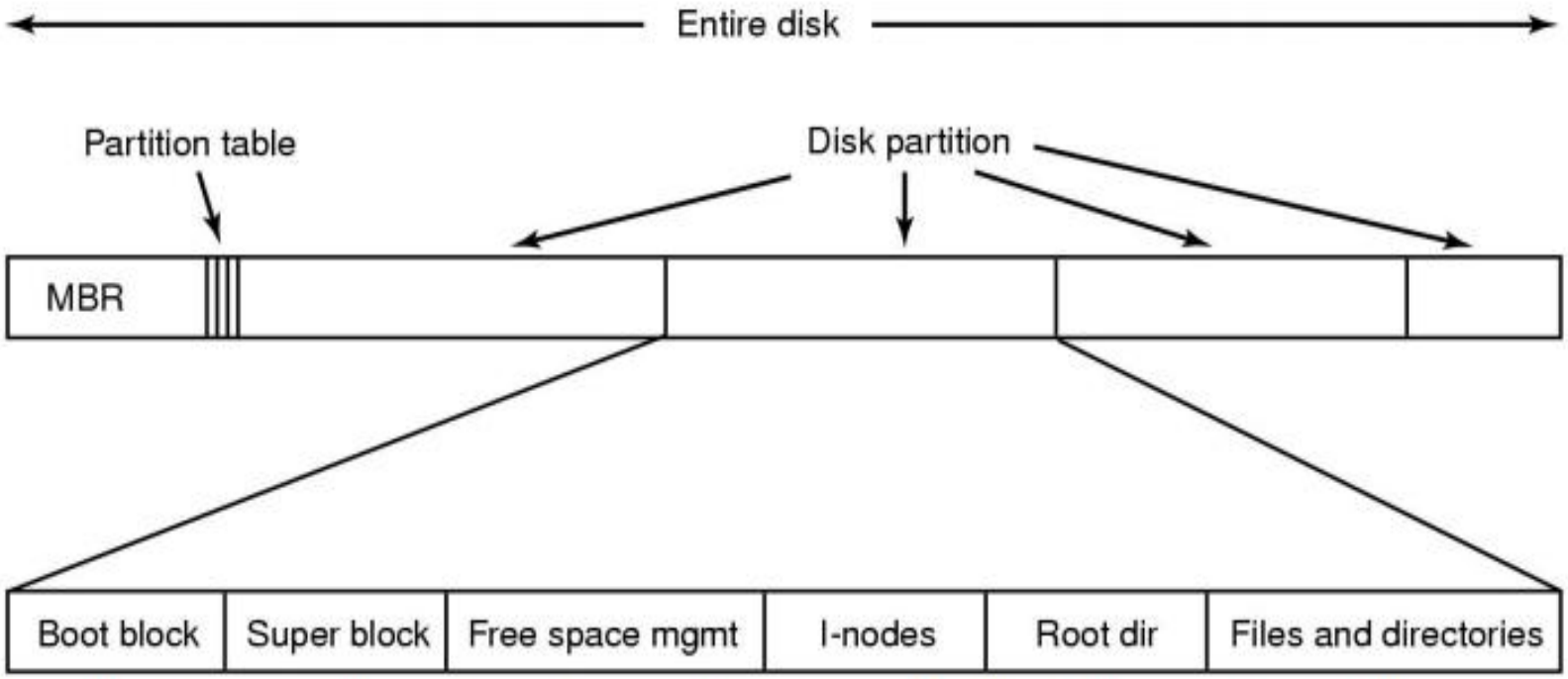
To allow to share the directory for different user acyclic-graph-structure is used.

Nearly all modern file systems are organized in this manner.



File-System Layout

Disks are divided up into one or more partitions, with independent file system on each partition.

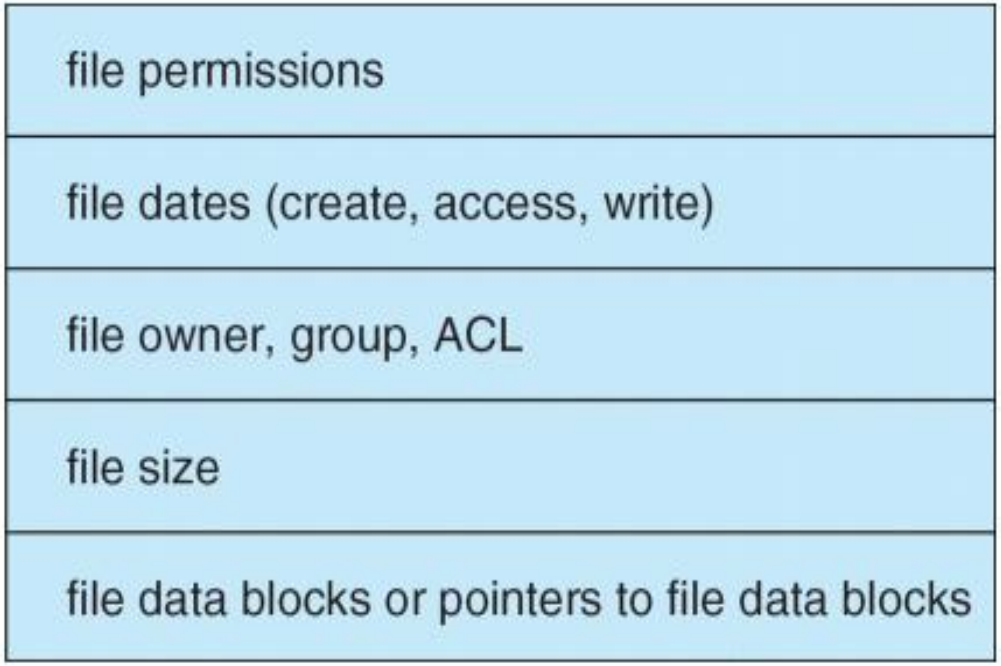


Boot Block: contain the information needed to boot OS from this partition, if this partition does not contain OS, this block will be empty.

Super block (with Free space mgmt.): Contain the partition detail, e.g number of block in the partition, size of block, free block count, free block pointer etc.

File Control Block (FCB)

FCB (also called inode) contains many details about the file, including the file permissions, ownership, size and location of data block.



To create a new file, file system allocates a new FCB. The system then reads the appropriate directory into memory, update it with the new file name and FCB, and writes it back to the disk.

Directory Implementation

The directory entry provides the information needed to find the disk block of the file. The file attributes are stored in the directory.

Linear List

Use the linear list of the file names with pointer to the data blocks.
It requires linear search to find a particular entry.
Advantages: Simple to implement.
Problem: linear search to find the file.

Hash Table

It consist linear list with hash table.
The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
If that key is already in use, a linked list is constructed.
Advantages: greatly decrease the file search time.
Problem: It greatly fixed size and dependence of the hash function on that size.

Allocation Methods

Contiguous Allocation

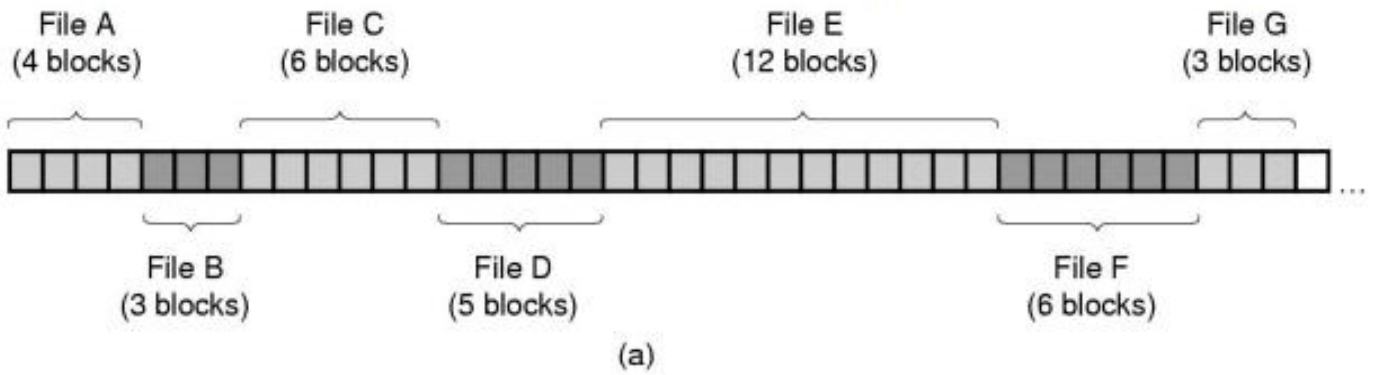
Each file occupy a set of contiguous block on the disk.

Disk addresses define a linear ordering on the disk.

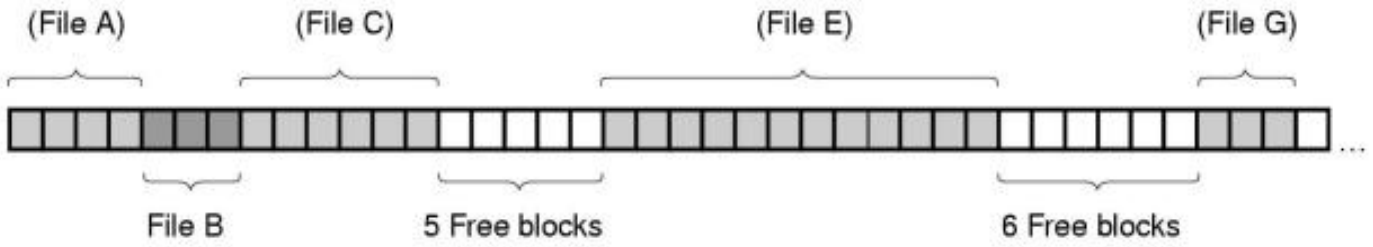
File is defined by the disk address and length in block units.

With 2-KB blocks, a 50-KB file would be allocated 25 consecutive blocks.

Both sequential and direct access can be supported by contiguous allocation.



(a)



(b)

Allocation Methods

Contiguous Allocation

Advantages:

Simple to implement; accessing a file that has been allocated contiguously is easy.

High performance; the entire file can be read in single operation i.e. decrease the seek time.

Problems:

fragmentation: when files are allocated and deleted the free disk space is broken into holes.

Dynamic-storage-allocation problem: searching of right holes.

Required pre-information of file size.

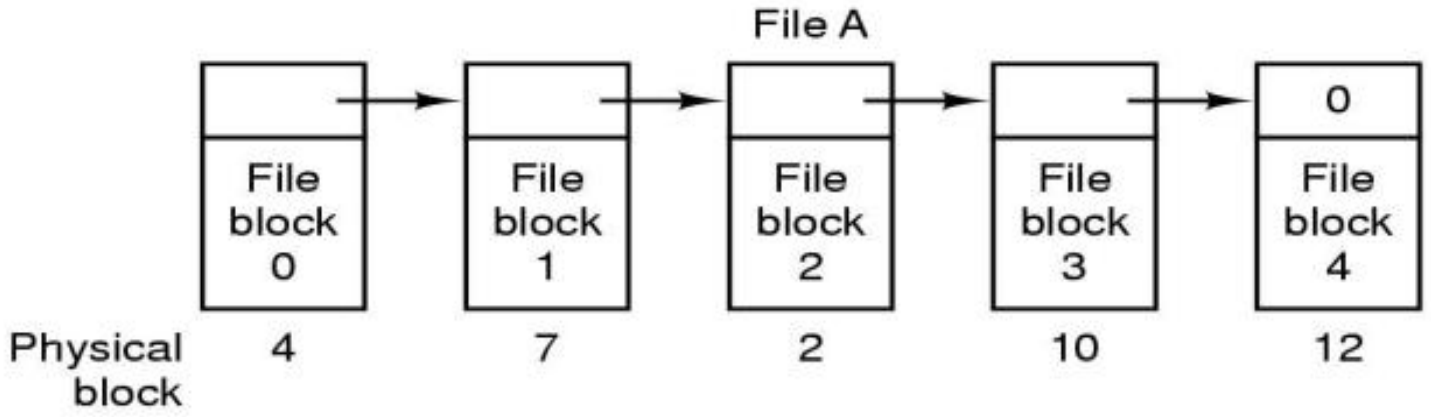
Due to its good performance it used in many system; it is widely used in CD-ROM file system.

Allocation Methods

Linked Allocation

Each file is a linked list of disk blocks; the disk block may be scattered anywhere on the disk.

Each block contain the pointer to the next block of same file. To create the new file, we simply create a new entry in the directory; with linked allocation, each directory entry has a pointer to the first disk block of the file.



Allocation Methods

Linked Allocation

Problems:

It solves all problems of contiguous allocation but it can used only for sequential access file; random access is extremely slow.

Each block access required disk seek.

It also required space for pointer.

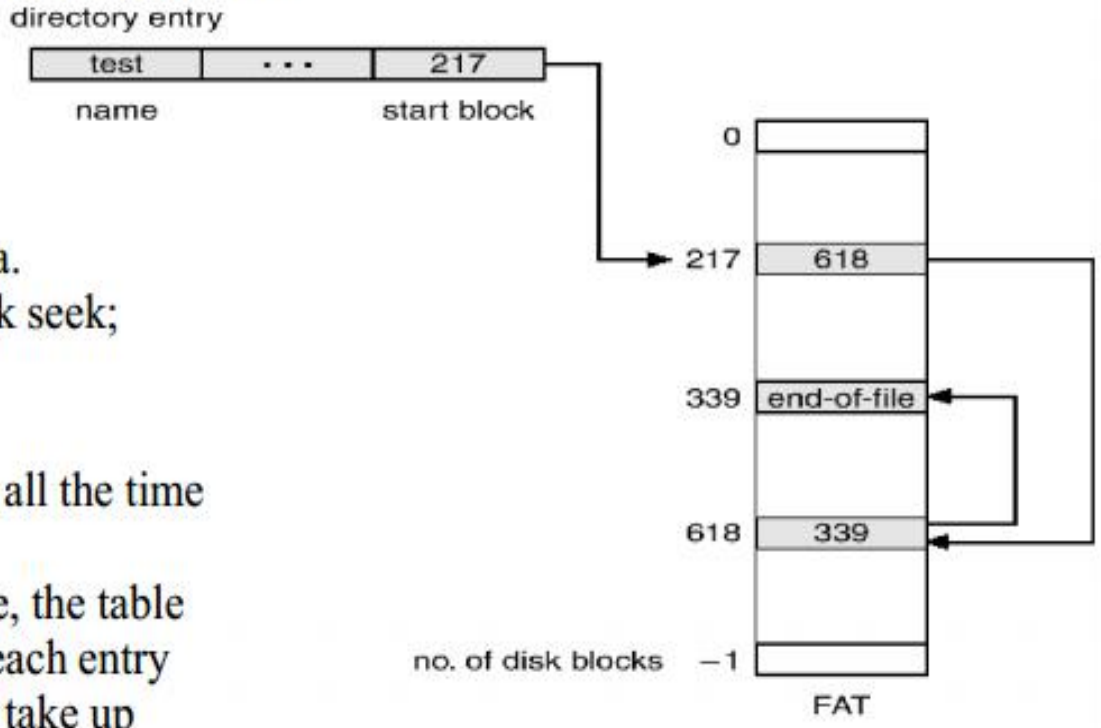
Solution: Using File Allocation Table (FAT).

The table has one entry for each disk block containing next block number for the file. This resides at the beginning of each disk partition.

Allocation Methods

Linked Allocation using FAT

The FAT is used as is a linked list. The directory entry contains the block number of the first block of the file. The FAT is looked to find next block until a special end-of-file value is reached.



Advantages:

The entire block is available for data.
Result the significant number of disk seek; random access time is improved.

Problems:

The entire table must be in memory all the time to make it work.
With 20GB disk and 1KB block size, the table needs 20 millions entries. Suppose each entry requires 3 bytes. Thus the table will take up 60MB of main memory all the time.

Allocation Methods

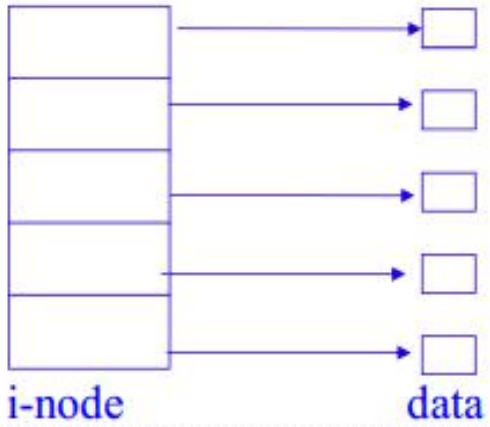
Index Allocation

To keep the track of which blocks belongs to which file, each file has data structure (i-node) that list the attributes and disk address of the disk block associate with the file.

Each i-node are stored in a disk block, if a disk block is not sufficient to hold i-node it can be multileveled.

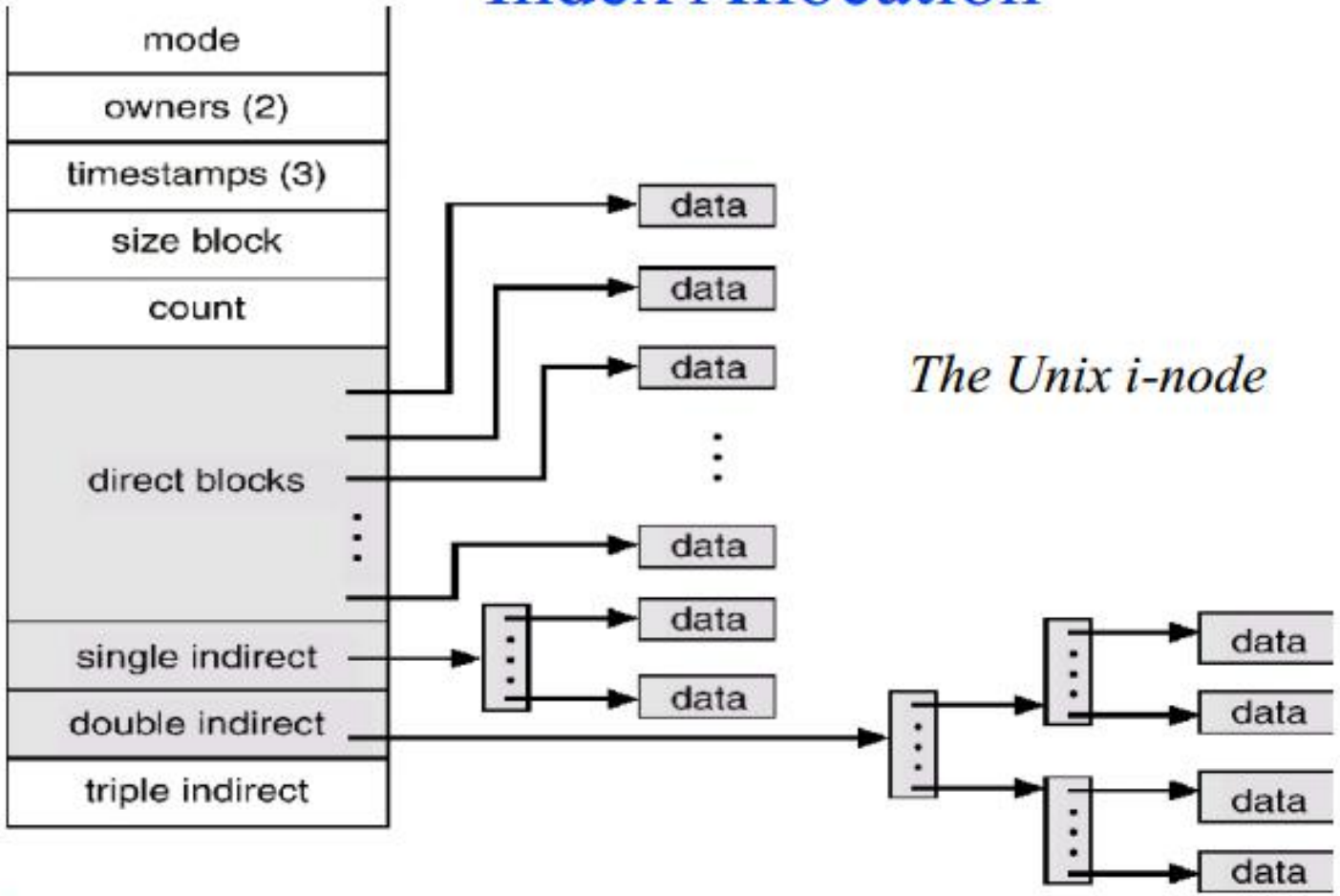
Independent to disk size.

If i-node occupies n bytes for each file and k files are opened, the total memory by i-nodes is kn bytes.



Allocation Methods

Index Allocation



The Unix i-node

Advantages: Far smaller than space occupied by FAT.

Problems: This can be suffer from performance.

System Protection

Reading: Chapter14 of Textbook

How to control the access of programs, processes, or users to the resources defined by a computer system?

How to ensure that each object is accessed correctly and only by those processes that are allowed to do so?

How to distinguish between authorized and unauthorized usage?

The solution to all these issues is to maintain the system protection.

Protection Principles

A key, guiding principle for protection is the – *principle of least privilege* - Programs, users and systems should be given just enough *privileges* to perform their tasks

- Limits damage if entity has a bug, gets abused (misused)
- Can be static (during life of system, during life of process) or dynamic (changed by process as needed) – **domain switching, privilege escalation**

Must consider "grain" aspect

Rough-grained privilege management easier, simpler, but least privilege now done in large chunks

For example, traditional Unix processes either have abilities of the associated user, or of root

Fine-grained management more complex, more overhead, but more protective

Access Matrix

- View protection as a matrix (*access matrix – a conceptual representation*)
- Rows represent domains and Columns represent objects
- Each entry, *Access(i, j),* is set of operations that process executing in $Domain_i$ can invoke on $Object_j$

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix

User who creates object can define access column for that object

Example: Access matrix with domains as objects.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Implementation Issues of Access Matrix

At run-time…

- What does the OS know about the user?
- What does the OS know about the resources?

What is the cost of checking and enforcing?

- Access to the data
- Cost of searching for a match

Impractical to implement full Access Matrix

- Size
- Access controls disjoint from both objects and domains

Implementation of Access Matrix

Generally, a sparse matrix, i.e. most of the entries will be empty.

There are several methods of implementation of matrix

Option 1 – Global table – simplest method

Store ordered triples < *domain, object, rights-set* > in table. A requested operation M on object O_j within domain D_i -> search global table for < D_i, O_j, R_k > with M ∈ R_k, if triple is found, the operation is allowed to continue.

But table could be large -> won't fit in main memory

Difficult to group objects (consider an object that all domains can read)

Option 2 – Access lists for objects

Each column implemented as an access list for one object. Resulting per-object list consists of ordered pairs < *domain, rights-set* > defining all domains with non-empty set of access rights for the object

Easily extended to contain default set -> If operation M ∈ default set, also allow access

For efficiency, first check default set and check for access list.

Implementation of Access Matrix

Option 3 – Capability list for domains

Instead of object-based, list is domain based. *Capability list* for domain is list of objects together with operations allows on them

Object represented by its name or address, called a *capability*

Execute operation M on object O_j, process requests operation and specifies capability as parameter

- Possession of capability means access is allowed

Capability list associated with domain but never directly accessible by domain

- Rather, protected object, maintained by OS and accessed indirectly
- Like a "secure pointer"
- Idea can be extended up to applications

Option 4 – Lock-key

Compromise between access lists and capability lists

Each object has list of unique bit patterns, called *locks*

Each domain as list of unique bit patterns called *keys*

Process in a domain can only access object if domain has key that matches one of the locks

END