UNIT IV

File Access Methods & Directories

S.Rajarajan

APIII/CSE

School of Computing

SASTRA

Access Methods

- Files **store information**.
- When it is used, this information must be **accessed and read** into **computer memory**.
- The information in the file can be accessed in **several ways.**

1. Sequential Access

- The simplest access method is **sequential access.**
- Information in the file is processed in order, **one record after the other.**
- This mode of access is by far the **most common**; for example, **editors** and **compilers** usually access files in this fashion.
- Reads and writes make up the bulk of the operations on a file.
- A **read operation- read_next()** reads the **next portion** of the file and **automatically advances** a **file pointer**, which tracks the I/O location.
- Similarly, the **write operation- write_next() append**s to the end of the file (**EOF**) and advances to the end of the newly written material.

2. Direct Access

- Another method is **direct access (or relative access).**
- Here, a file is made up of **fixed-length logical records** that allow programs to read and write records rapidly in **no particular order.**
- The direct-access method is based on a **disk model of a file**, since disks allow **random access** to any file block.
- For direct access, the file is viewed as a **numbered** sequence of **blocks or records**.

- Thus, we may **read block 14**, then **read block 53**, and then **write block 7**.
- There are **no restrictions** on the **order** of reading or writing for a direct-access file.
- Direct-access files are of great use for **immediate access to large amounts** of information.
- **Databases** are often of this type.
- For the direct-access method, the file operations must be modified to include the **block number** as a **parameter.**
- Thus, we have **read(n),** where **n** is the **block number.**

- The **block number** provided by the user to the operating system is normally a **relative block number.**
- A relative block number is an **index** relative to the **beginning of the file**.
- Thus, the first relative block of the file is 0, the next is 1, and so on.
- The use of relative block numbers allows the operating system to decide where the file should be placed (called the **allocation problem)** and helps to **prevent the user** from **accessing portions of the file** system that **may not be part of his file**

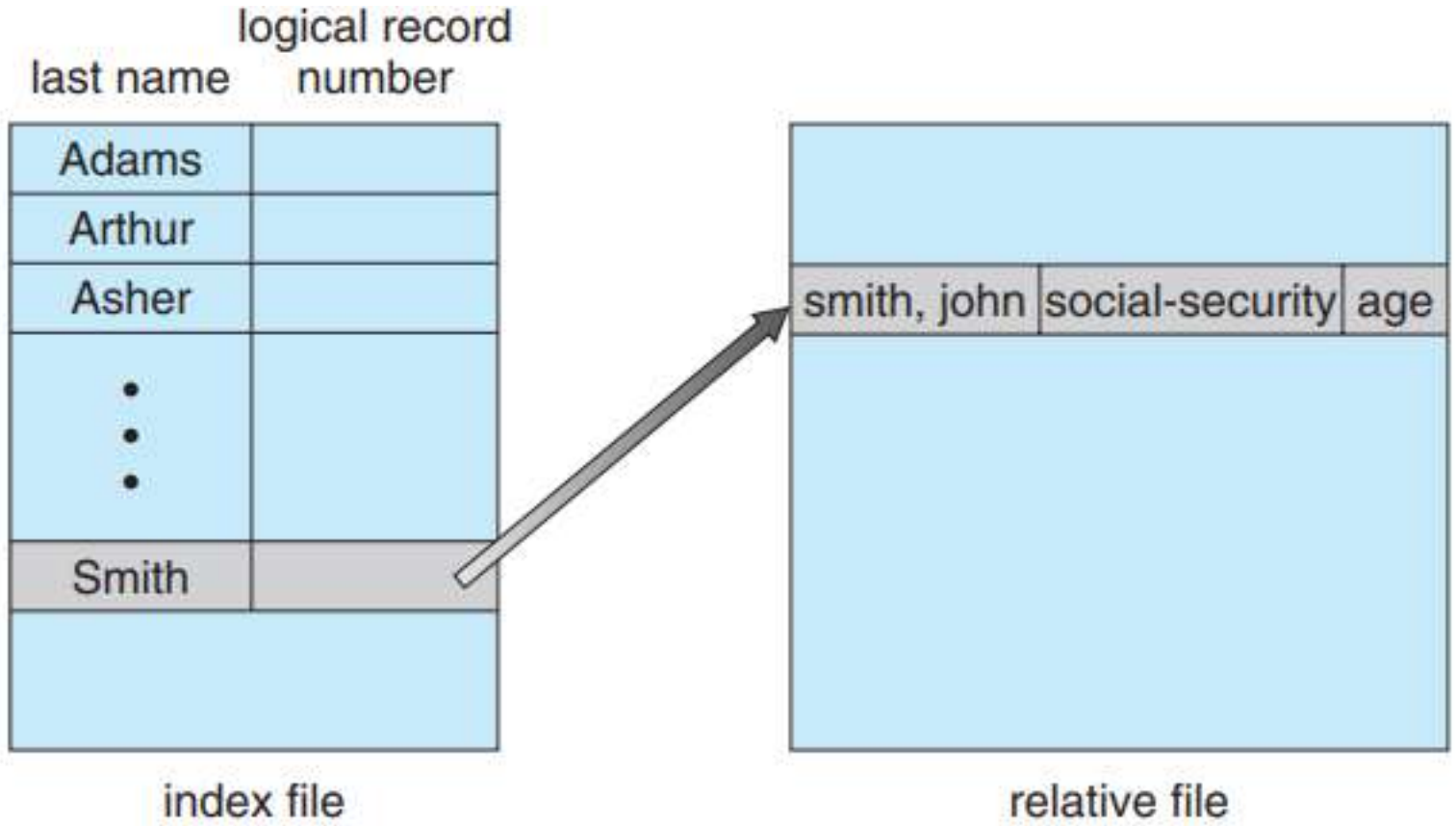
- Not all operating systems support both sequential and direct access for files.
- Some systems allow only sequential file access; others allow only direct access.

3. Indexed sequential access

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.
- The **index,** like an index in **the back of a book**, contains **pointers** to the various blocks.
- To find a record in the file, we **first search the index** and then **use the pointer** to access the file **directly** and to find the **desired record**

- With large files, the **index file** itself may become **too large** to be kept in memory.
- One solution is to create an **index for the index file**.
- The **primary index file** contains **pointers to secondary index files**, which point to the **actual data items**.
- IBM's indexed sequential-access method (ISAM) uses a small **master index** that points **to disk blocks of a secondary index**.
- The **secondary index blocks** point to the **actual file blocks**.
- The file is kept **sorted on a defined key**.
- To find a particular item, we first make a **binary search** of the **master index**, which provides the **block number** of the **secondary index**.

- This block is read in, and again a **binary search** is used on **secondary index** to find the **block** containing the **desired record.**

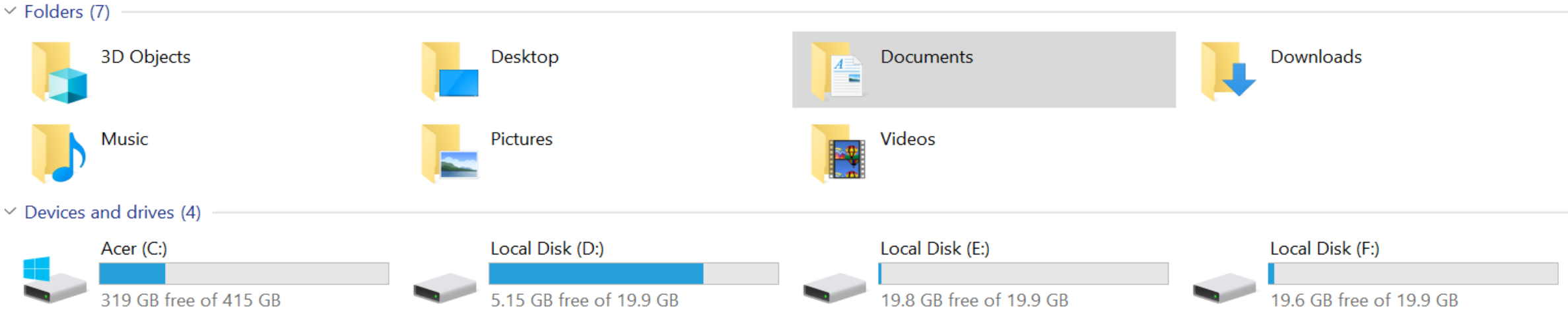

logical record
last name
number
Adams
Arthur
Asher
Smith
smith, john
social-security
age
index file
relative file

Directory and Disk Structure

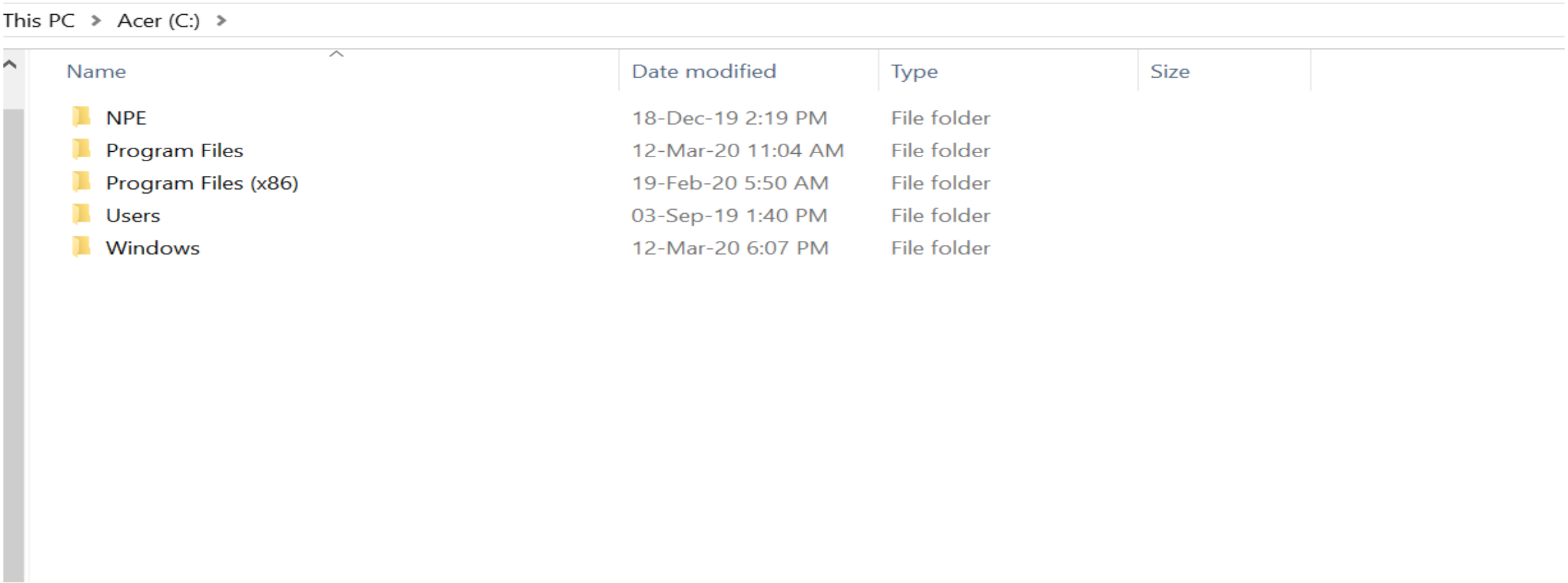
- Next, we consider **how to store files.**
- There are typically **thousands, millions, even billions of files** within **a computer**.
- Files are stored on r**andom-access storage devices**, including **hard disks**, **optical disks**, and **solid-state** (memory-based) disks.
- A storage device can be used in its **entirety for a file system** or It can also **be subdivided for finer-grained control.**
- For example, a disk can be **partitioned** into quarters, and each quarter can hold a separate file system.

- Storage devices can also be collected together into **RAID sets** that provide **protection from the failure** of a single disk.
- Partitioning is useful for
  - **limiting the sizes of individual file systems**
  - **putting multiple file-system types on the same device**,
  - **Or leaving part of the device available for other uses**, such **as swap space or unformatted** (raw) disk space.
- A file system can be created on each of these parts of the disk.
- Any entity containing a file system is generally known as a **volume**

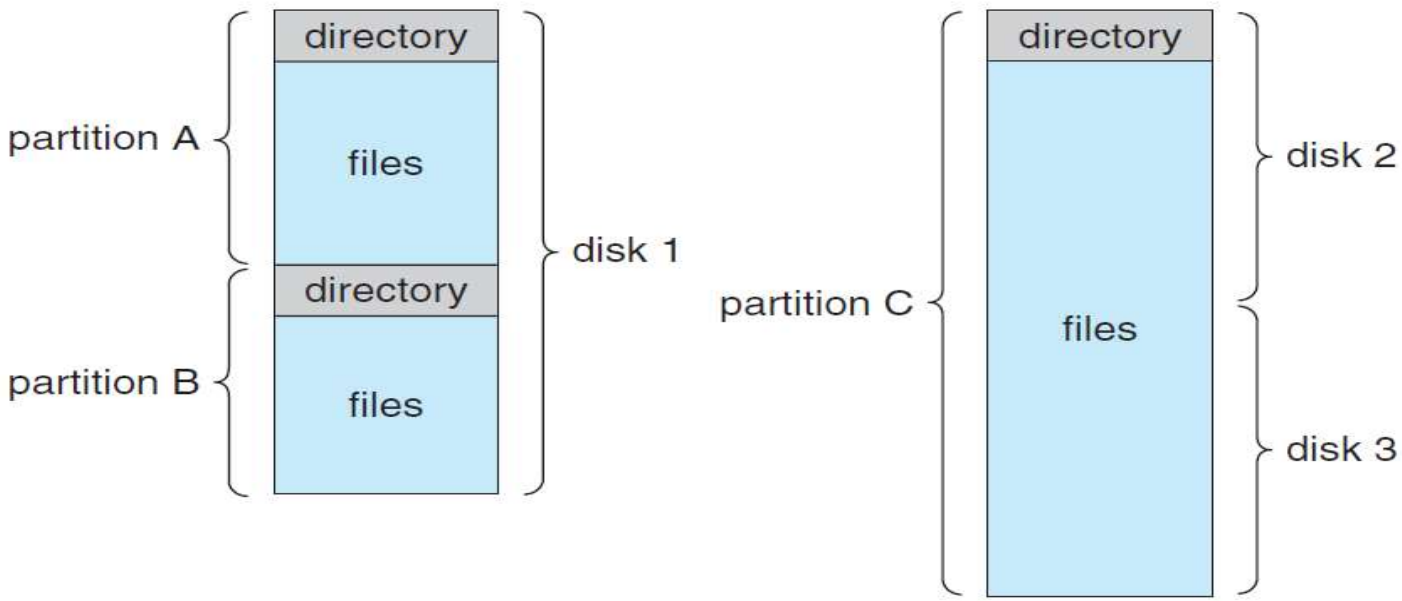
Disk Volumes on Windows



- Each volume can be thought of as **a virtual disk**.
- Volumes can also store **multiple operating systems**, allowing a system to **boot** and run more than one operating system.
- Each volume that **contains a file system** must also contain information about the **files in the system**.
- This information is kept in entries in a **device directory or volume table of contents**


This PC > Acer (C:) >
Name
Date modified
Type
Size
NPE
18-Dec-19 2:19 PM
File folder
Program Files
12-Mar-20 11:04 AM
File folder
Program Files (x86)
19-Feb-20 5:50 AM
File folder
Users
03-Sep-19 1:40 PM
File folder
Windows
12-Mar-20 6:07 PM
File folder

Directories



Directory

/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs

Directory Overview

- The directory can be viewed as a **symbol table** that **translates file names** into their **directory entries.**
- The directory itself can be **organized in many ways.**
- The organization must allow us **to insert entries**, to **delete entries**, to **search for a named entry,** and to **list all the entries** in the directory.

- The operations that are to be performed on a directory:
- **Search for a file.** We need to be able to search a directory structure.
- **Create a file.** New files need to be created and added to the directory.
- **Delete a file.** When a file is no longer needed, we want to be able to remove it from the directory.
- **List a directory.** We need to be able to list the files in a directory.
- **Rename a file.** Change the name of a file
- **Traverse the file system.** We may wish to access every directory and every file within a directory structure

Single-Level Directory

- The simplest directory structure is the single-level directory.
- All files are contained in the same directory, which is easy to support and understand.

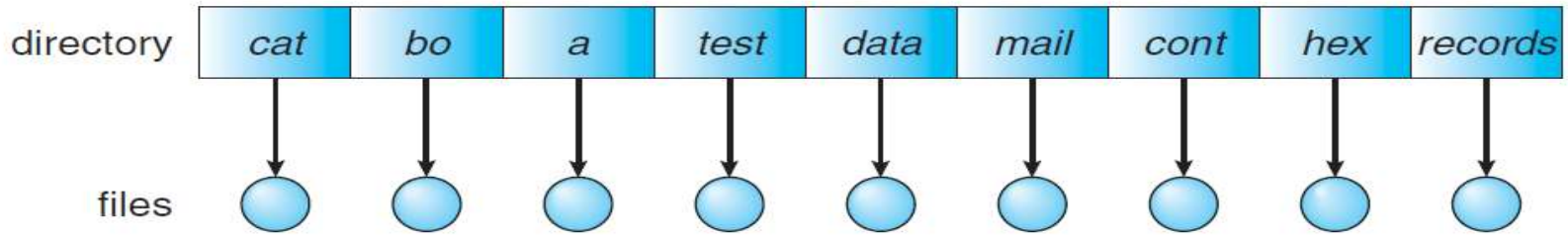
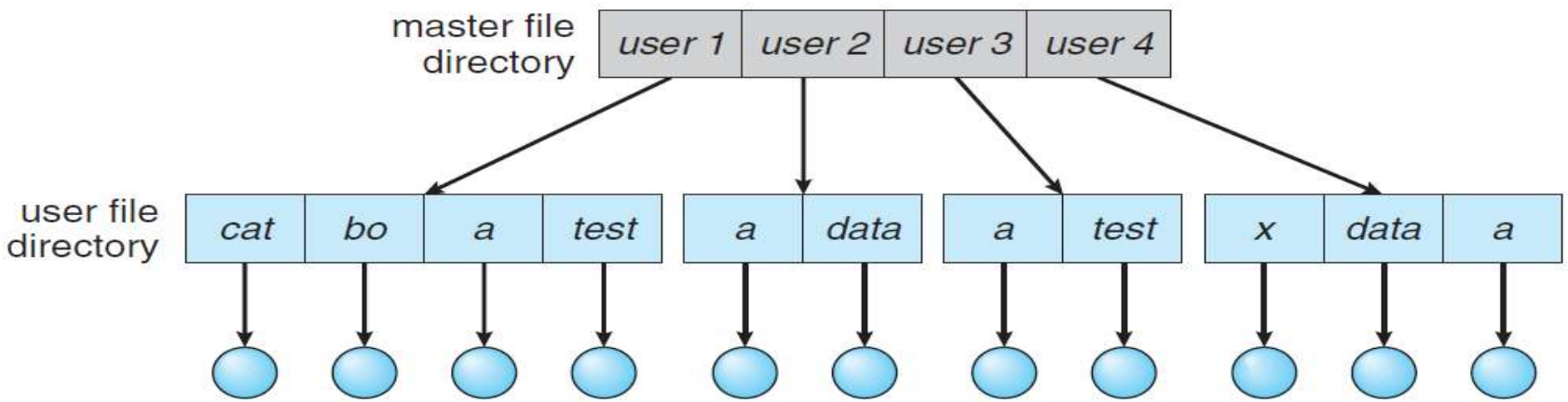


Figure 11.9 Single-level directory.

Two-Level Directory

- Suitable for multi-user systems.
- In the two-level directory structure, each user has his own **user file directory (UFD).**
- When a user job starts or a user logs in, the system's **master file directory (MFD) is searched.**
- The MFD is **indexed by user name** or account number, and each entry **points to the UFD** for that user.
- When a user refers to a particular file, only his own UFD is searched.

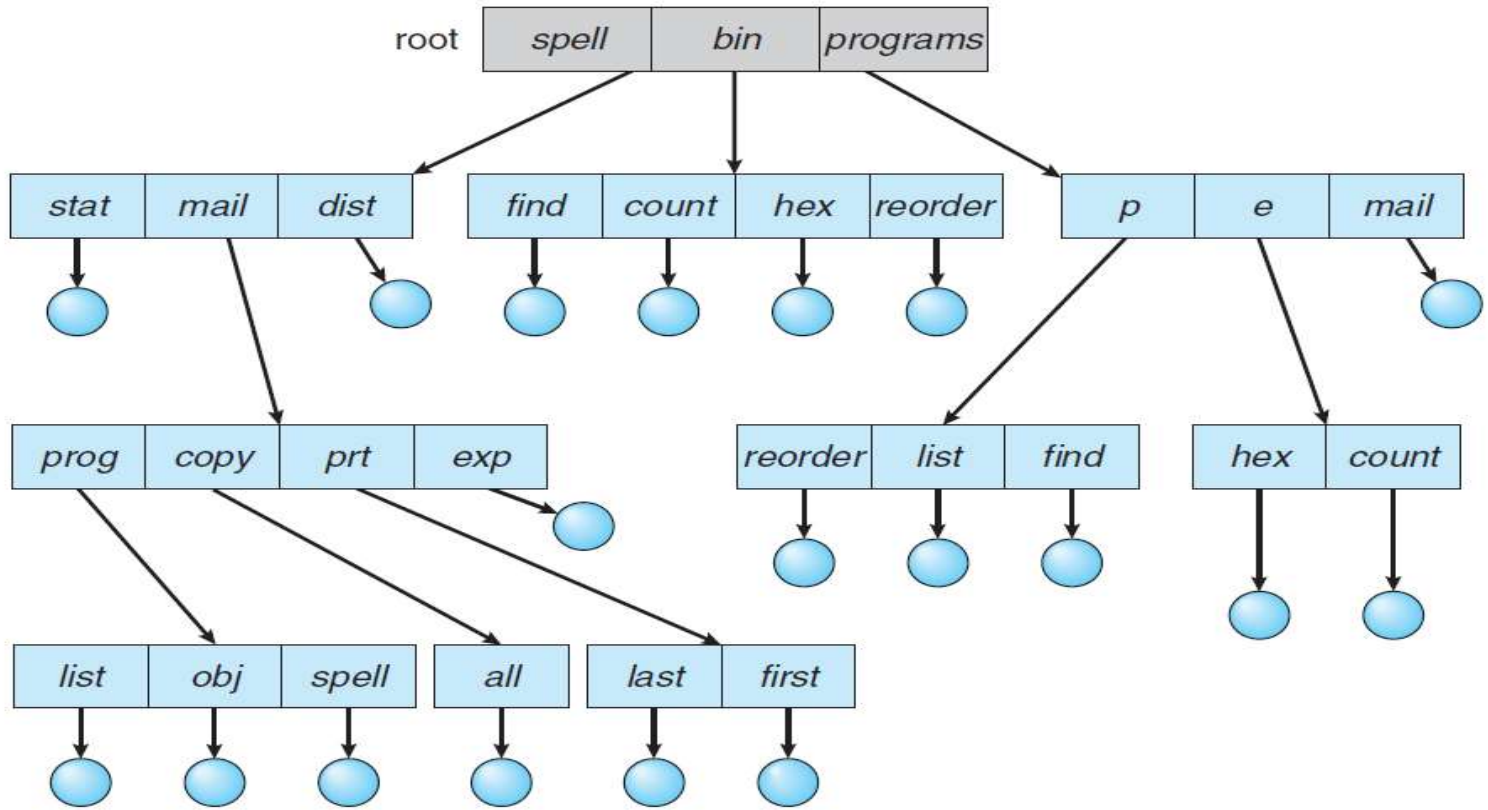

master file directory
user 1
user 2
user 3
user 4
user file directory
cat
bo
a
test
a
data
a
test
x
data
a

Tree-Structured Directories

- The natural generalization is to extend the directory structure to a tree of arbitrary height.
- Allows users **to create their own subdirectories** and to organize their files accordingly.
- A tree is the most common directory structure.
- The tree has a **root directory**, and every file in the system has a **unique path name**.
- A **directory** (or subdirectory) contains a **set of files or subdirectories.**
- A directory is simply another file, but it is treated in a special way.

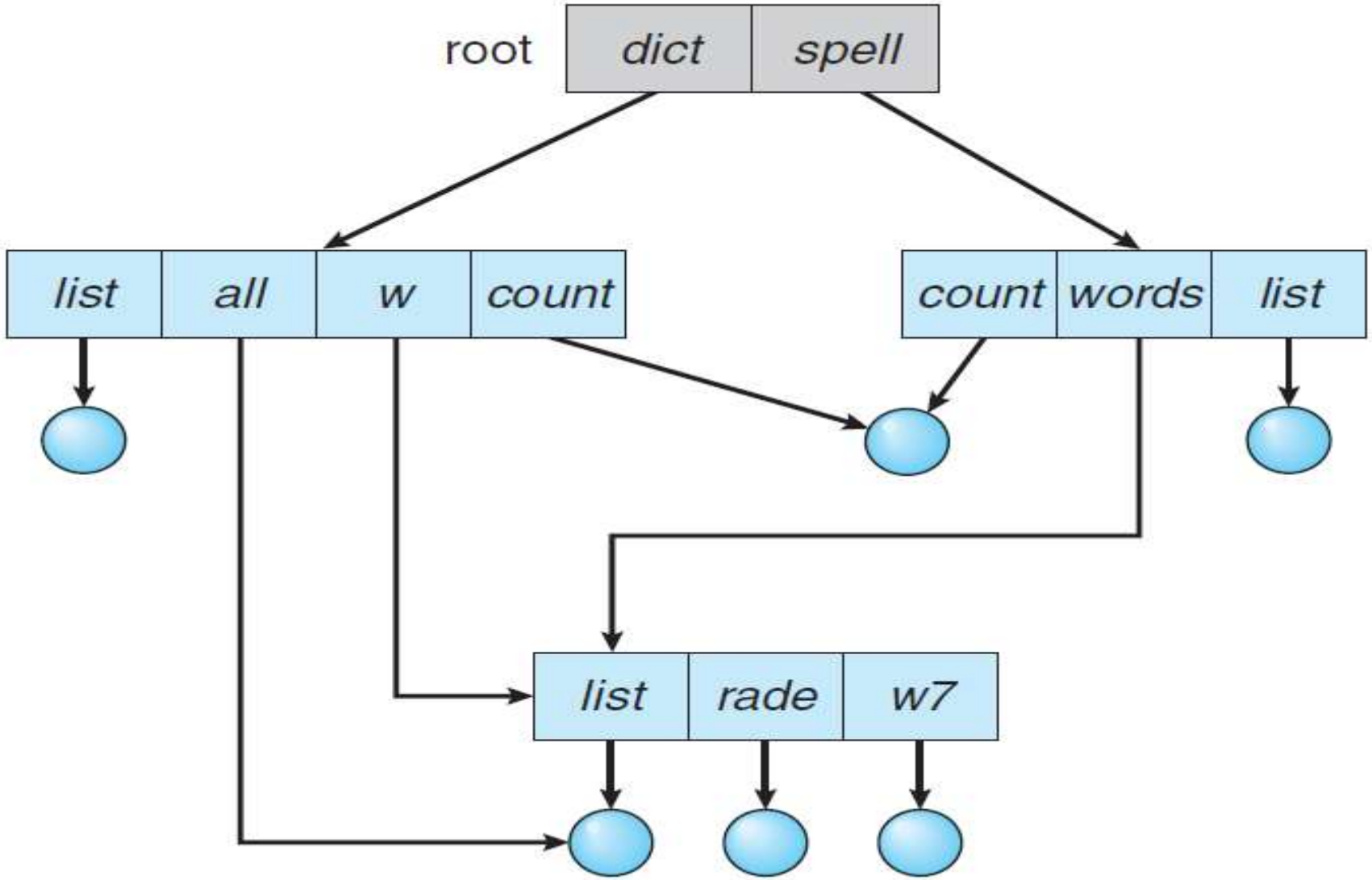
- Special **system calls** are used to **create and delete** directories.
- In normal use, each process has a **current directory.**
- The **current directory** should contain most of the files that are of current interest to the process.
- When reference is made to a file, the **current directory** is **searched**.
- If a file is needed that is **not in the current** directory, then the user usually must either specify a **path name** or **change the current directory** to be the directory holding that file.

- Path names can be of two types: absolute and relative.
- An **absolute path name** begins **at the root** and follows **a path** down to the specified file, giving the directory names on the path.
- A **relative path name** defines a path from the **current directory.**
- With a tree-structured directory system, users can be allowed to access, in addition to their files, the files of other users.


root
spell
bin
programs
stat
mail
dist
find
count
hex
reorder
p
e
mail
prog
copy
prt
exp
reorder
list
find
hex
count
list
obj
spell
all
last
first

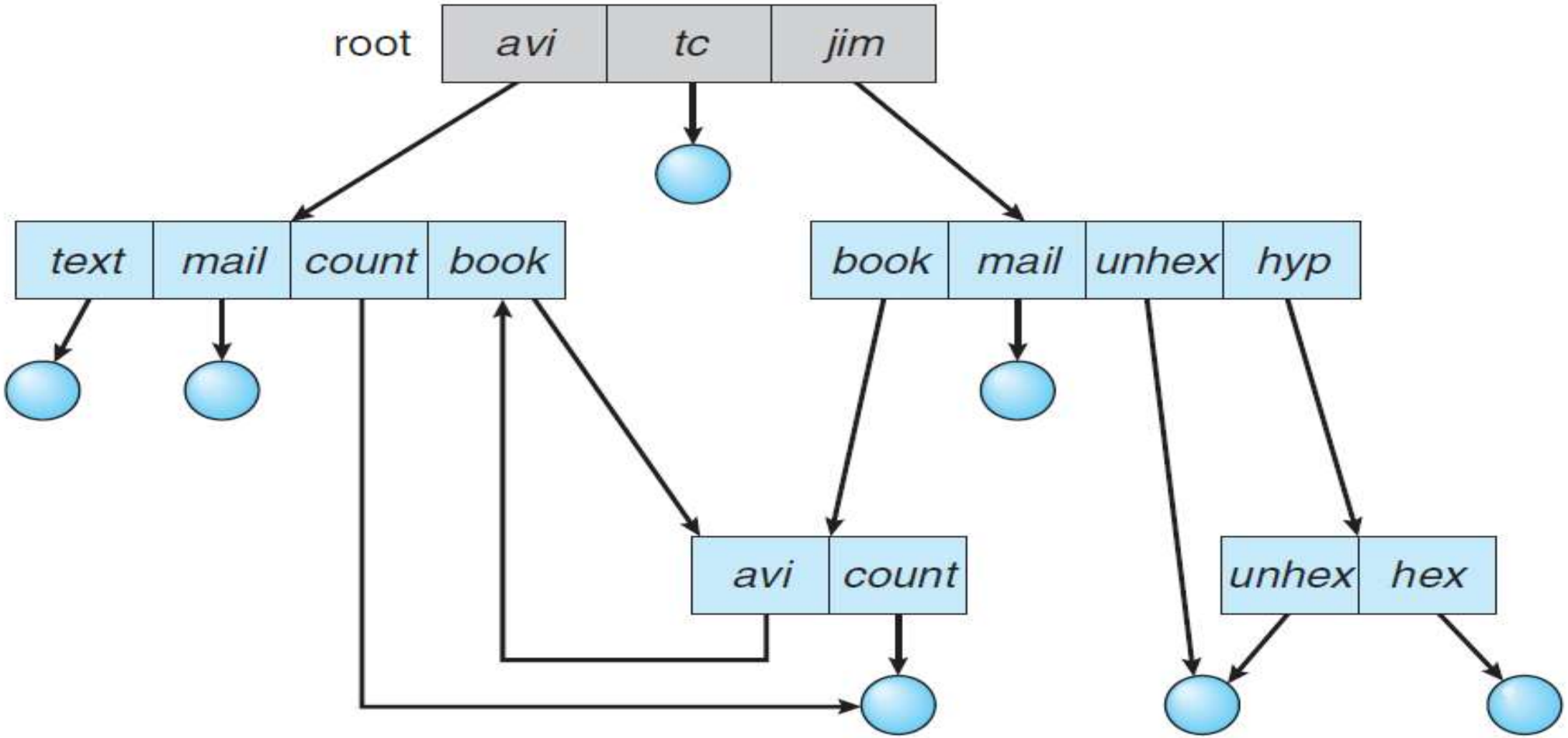
Acyclic-Graph Directories

- A tree structure prohibits the sharing of files or directories.
- An **acyclic graph** —that is, a graph with no cycles—allows directories to share subdirectories and files.
- The **same file or subdirectory** may be in **two different directories.**
- The acyclic graph is a natural generalization of the tree-structured directory scheme.
- It is important to note that a shared file (or directory) is **not the same as two copies** of the file.


root
dict
spell
list
all
w
count
count
words
list
list
rade
w7

General Graph Directory

- A **serious problem** with using an acyclic-graph structure is **ensuring that** there are **no cycles**.
- If we start with a two-level directory and allow users to create subdirectories, a **tree-structured directory** results.
- Simply **adding new files** and **subdirectories** to an existing tree-structured directory **preserves the tree-structured** nature.
- However, when we **add links**, the tree structure is destroyed, resulting in a **simple graph structure** which **may contain cycles**


root
avi
tc
jim
text
mail
count
book
book
mail
unhex
hyp
avi
count
unhex
hex