# Introduction to File Systems

Adapted form Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc.

# File Systems

- Standard of organizing data in disks

# File Systems

- Many important applications need to store more information than have in virtual address space of a process
- The information must survive the termination of the process using it.
- Multiple processes must be able to access the information concurrently.

# File Systems

- Disks are used to store files
- Information is stored in blocks on the disks
- Can read and write blocks

# File Systems

- Use file system as an abstraction to deal with accessing the information kept in blocks on a disk
- Files are created by a process
- Thousands of them on a disk
- Managed by the OS

# File Systems

- OS structures them, names them, protects them
- Two ways of looking at file system
  - User - how do we name a file, protect it, organize the files
  - Implementation - how are they organized on a disk

# File Systems

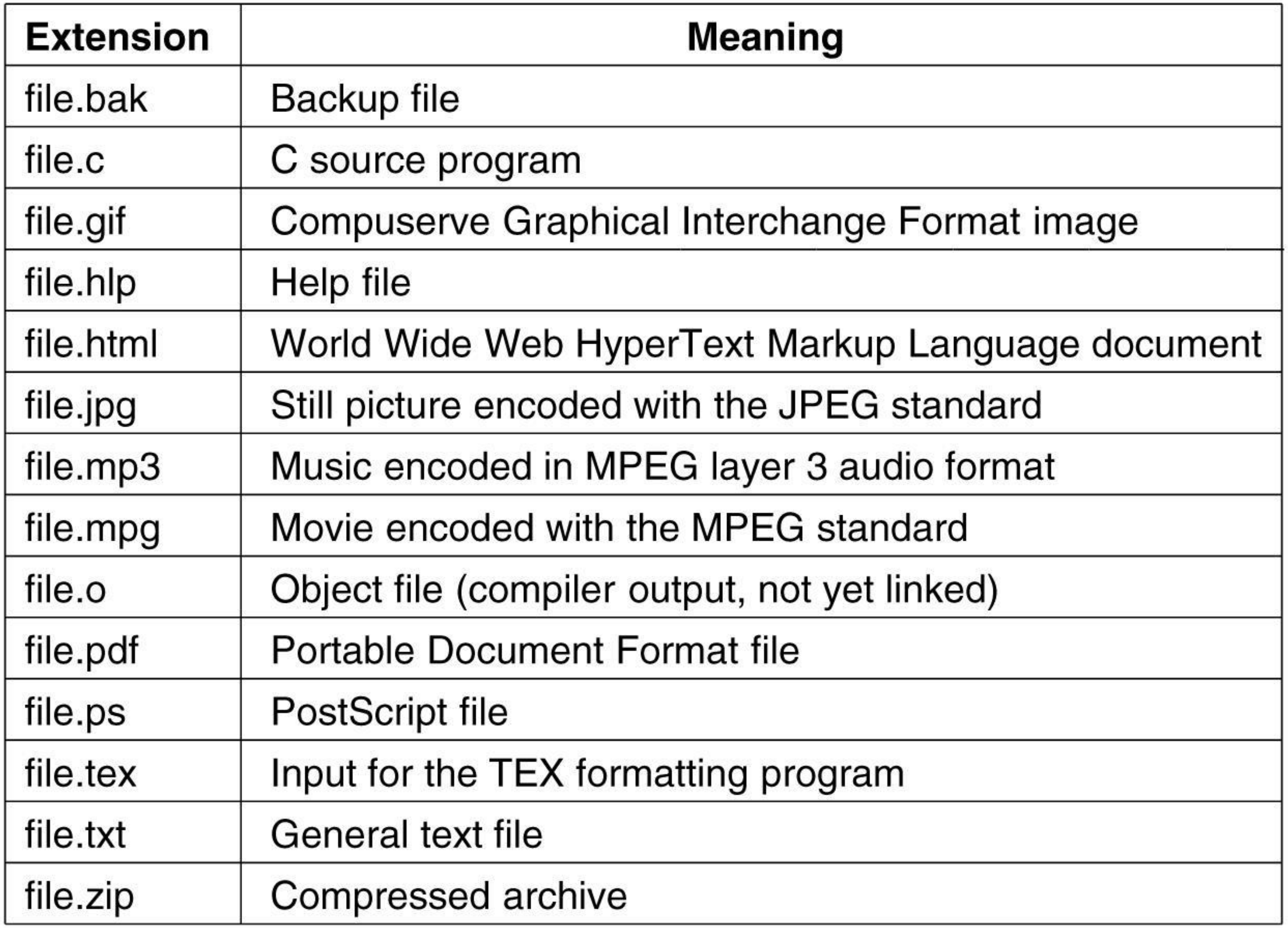
The user point of view

- Naming
- Structure
- Directories

# Naming

- FAT (16 and 32) were used in first Windows systems
- NTFS is used in latest Windows systems
- All OS’s use suffix as part of name
- UNIX does not always enforce a meaning for the suffixes
- DOS does enforce a meaning

# Suffix Examples



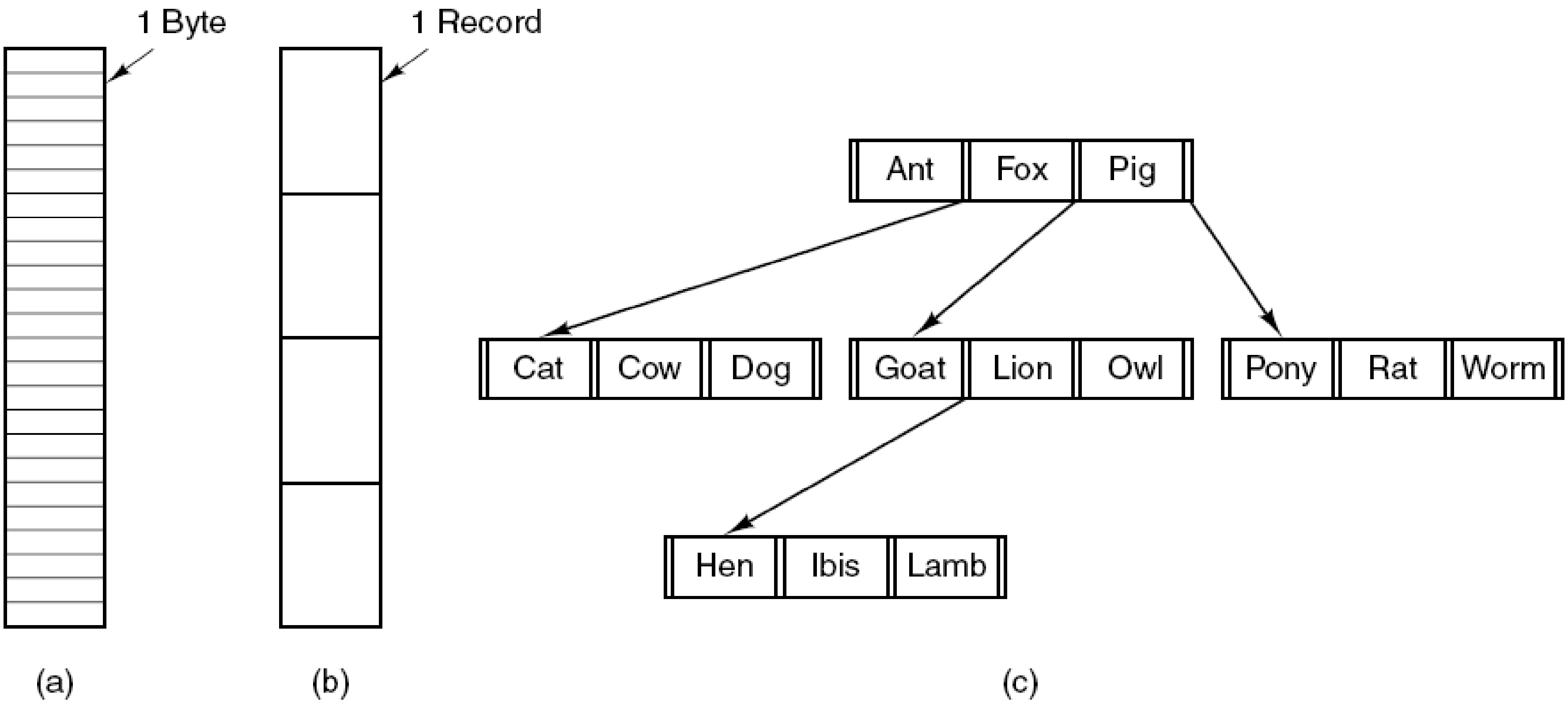
| Extension | Meaning |
|---|---|
| file.bak | Backup file |
| file.c | C source program |
| file.gif | Compuserve Graphical Interchange Format image |
| file.hlp | Help file |
| file.html | World Wide Web HyperText Markup Language document |
| file.jpg | Still picture encoded with the JPEG standard |
| file.mp3 | Music encoded in MPEG layer 3 audio format |
| file.mpg | Movie encoded with the MPEG standard |
| file.o | Object file (compiler output, not yet linked) |
| file.pdf | Portable Document Format file |
| file.ps | PostScript file |
| file.tex | Input for the TEX formatting program |
| file.txt | General text file |
| file.zip | Compressed archive |

.

# File Structure

- Byte sequences
  - Maximum flexibility - can put anything in
  - Unix and Windows use this approach
- Fixed length records
- Tree of records- uses key field to find records in the tree

# File Structure



Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

# File Types

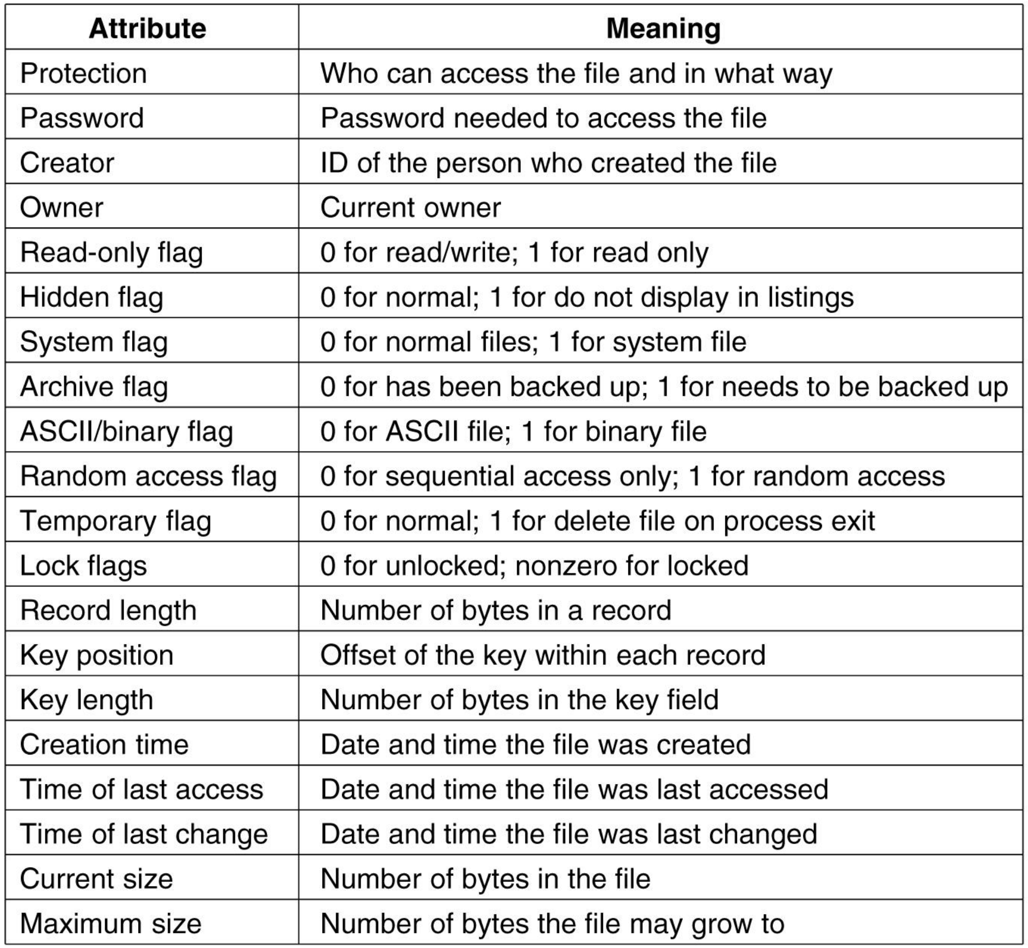
- Regular- contains user information

  Data files or Executable files
- Directories

# File Access

- Sequential access- read from the beginning, can’t skip around
  - Corresponds to magnetic tape
- Random access- start where you want to start
  - Came into play with disks
  - Necessary for many applications, e.g. airline reservation system

# File Attributes (hypothetical OS)



| Attribute | Meaning |
|---|---|
| Protection | Who can access the file and in what way |
| Password | Password needed to access the file |
| Creator | ID of the person who created the file |
| Owner | Current owner |
| Read-only flag | 0 for read/write; 1 for read only |
| Hidden flag | 0 for normal; 1 for do not display in listings |
| System flag | 0 for normal files; 1 for system file |
| Archive flag | 0 for has been backed up; 1 for needs to be backed up |
| ASCII/binary flag | 0 for ASCII file; 1 for binary file |
| Random access flag | 0 for sequential access only; 1 for random access |
| Temporary flag | 0 for normal; 1 for delete file on process exit |
| Lock flags | 0 for unlocked; nonzero for locked |
| Record length | Number of bytes in a record |
| Key position | Offset of the key within each record |
| Key length | Number of bytes in the key field |
| Creation time | Date and time the file was created |
| Time of last access | Date and time the file was last accessed |
| Time of last change | Date and time the file was last changed |
| Current size | Number of bytes in the file |
| Maximum size | Number of bytes the file may grow to |

# System Calls for files

- Create -with no data, sets some attributes
- Delete-to free disk space
- Open- after create, gets attributes and disk addresses into main memory
- Close- frees table space used by attributes and addresses
- Read-usually from current pointer  position. Need to specify buffer into which data is placed
- Write-usually to current position

# System Calls for Files

- Append- at the end of the file
- Seek-puts file pointer at specific place in file. Read or write from that position on
- Get Attributes-e.g. make needs most recent modification times to arrange for group compilation
- Set Attributes-e.g. protection attributes
- Rename

# How can system calls be used? An example-copyfile abc xyz

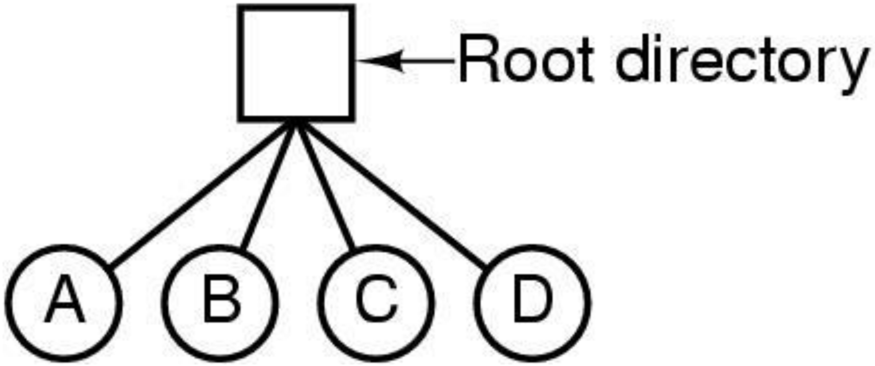
- Copies file abc to xyz
- Uses system calls (open, read, write, close)
- Reads and writes in 4K chunks
- Read (system call)  into a buffer
- Write (system call) from buffer to output file

# Directories

- Files which are used to organize a collection of files
- Also called folders in Windows

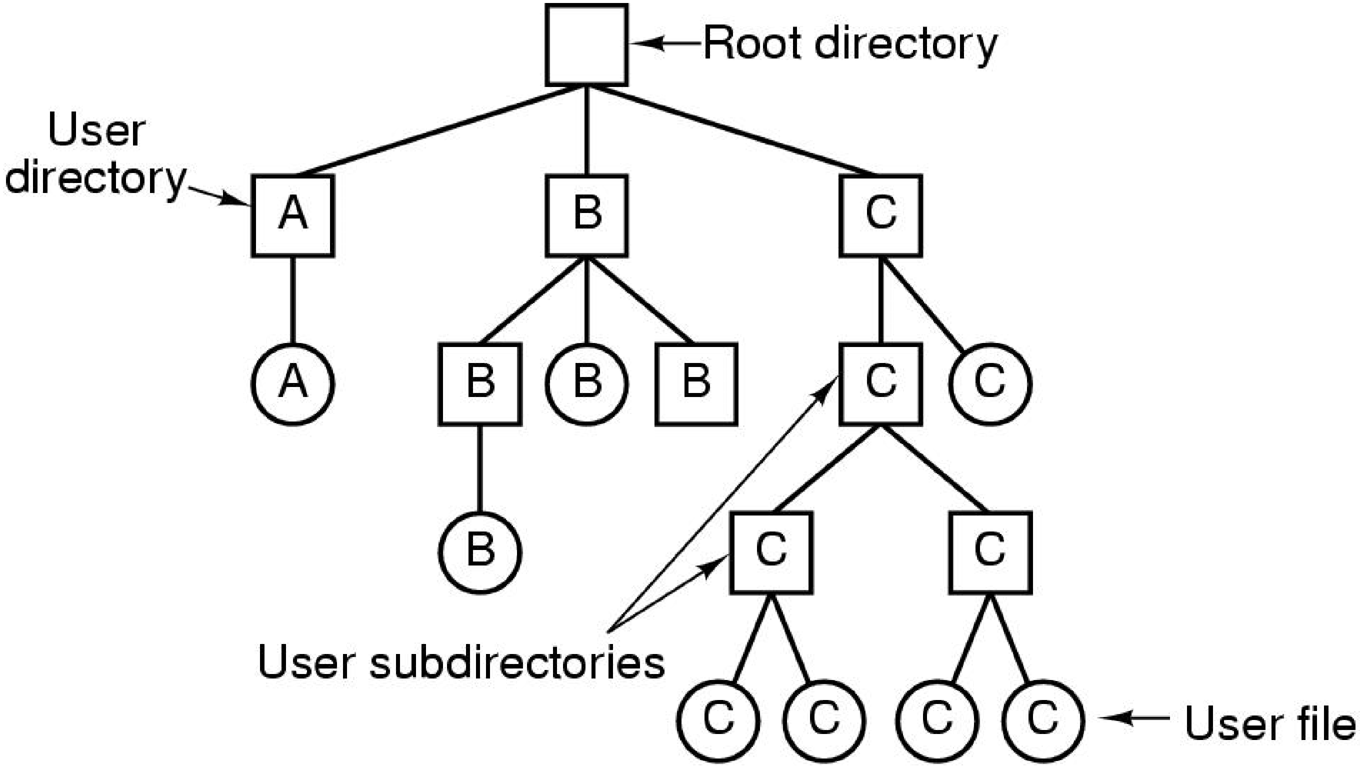
.

# Single Level Directory Systems



A single-level directory system containing four files.

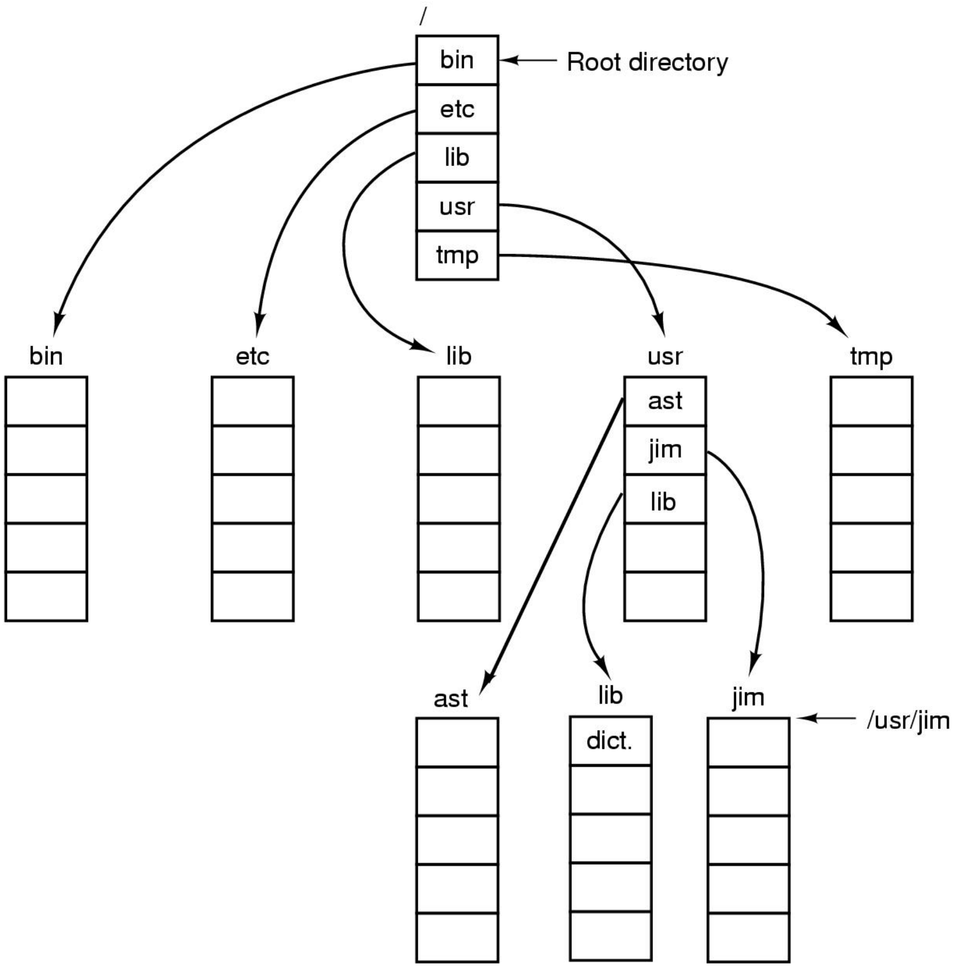
# Hierarchical Directory Systems



# Path names

- Absolute /usr/carl/cs310/miderm/answers
- Relative cs310/midterm/answers
- . Refers to current (working) directory
- .. Refers to parent of current directory

# Path Names



A UNIX directory tree.

# Directory Operations

- Create
- Delete
- Opendir
- Closedir
- Rename, …

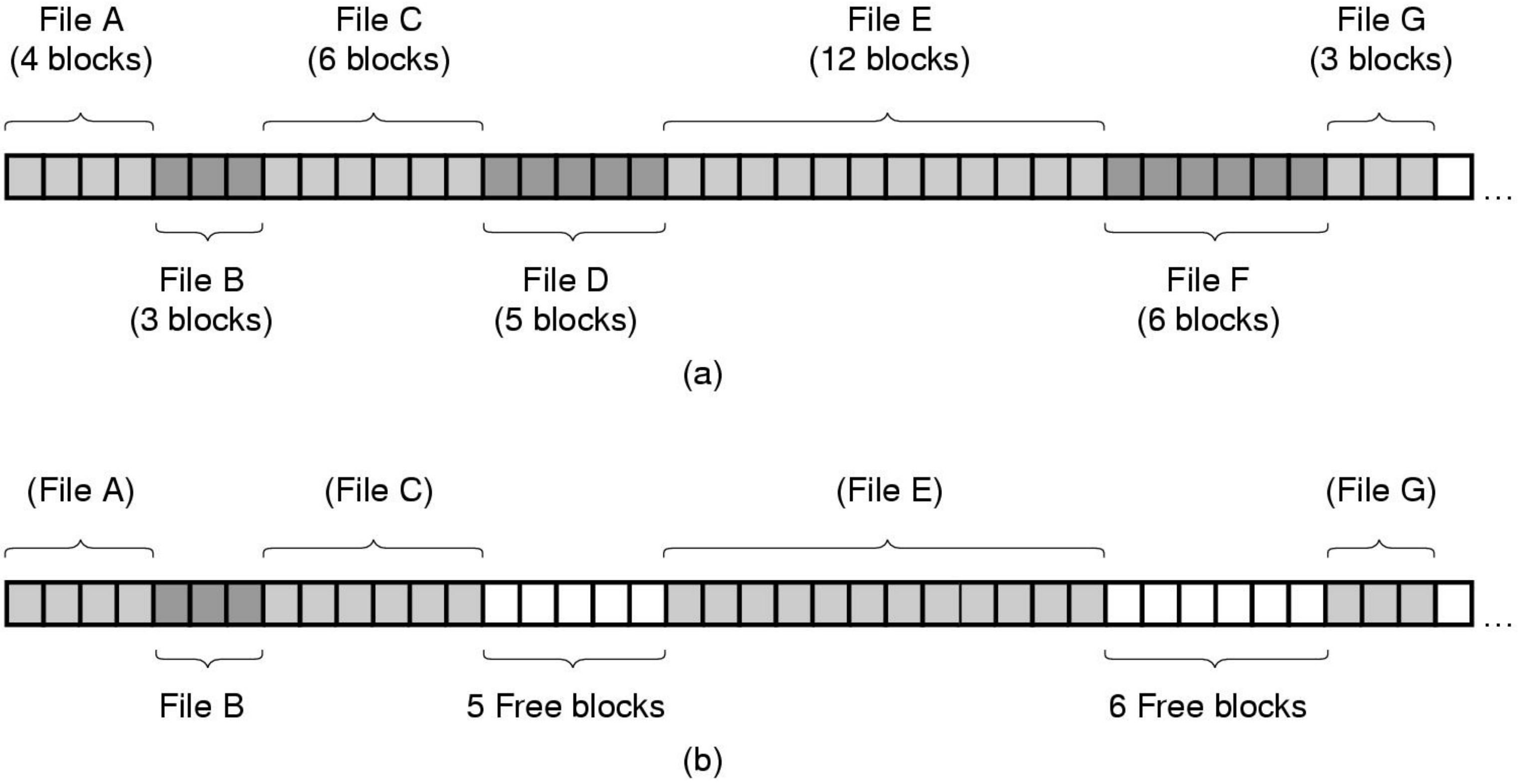
# File Implementation

- Files stored on disks.
- File blocks are placed on disk blocks

# Allocating Blocks to files

- Most important implementation issue
- Methods
  - Contiguous allocation
  - Linked list allocation
  - Linked list using table
  - I-nodes

# Contiguous Allocation



(a) Contiguous allocation of disk space for 7 files.

(b) The state of the disk after files D and F have been removed.

# Contiguous Allocation

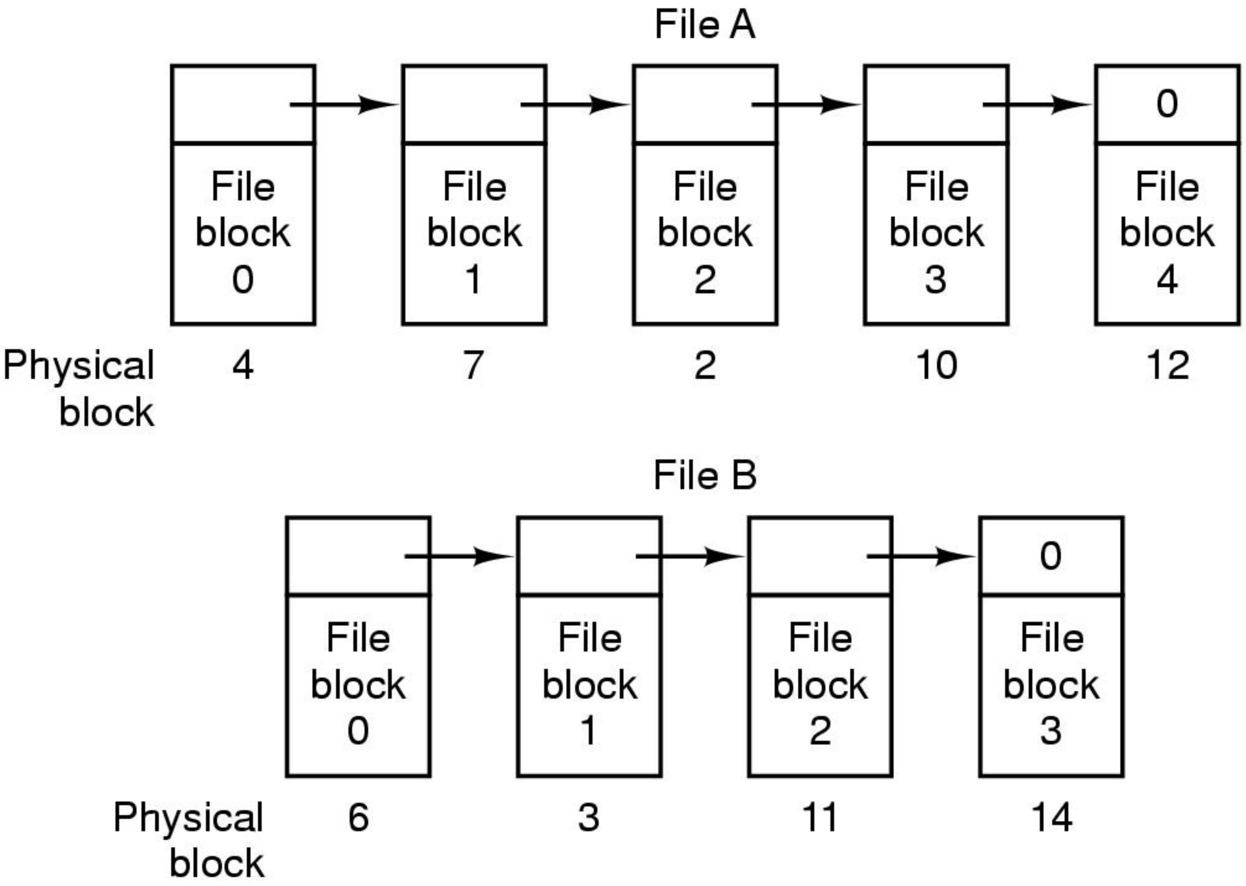
The good

- Easy to implement
- Read performance is great. Only need one seek to locate the first block in the file. The rest is easy.

The bad

- Disk becomes fragmented over time

# Linked List Allocation



**Storing a file as a linked list of disk blocks.**

# Linked Lists

The good

- Gets rid of fragmentation

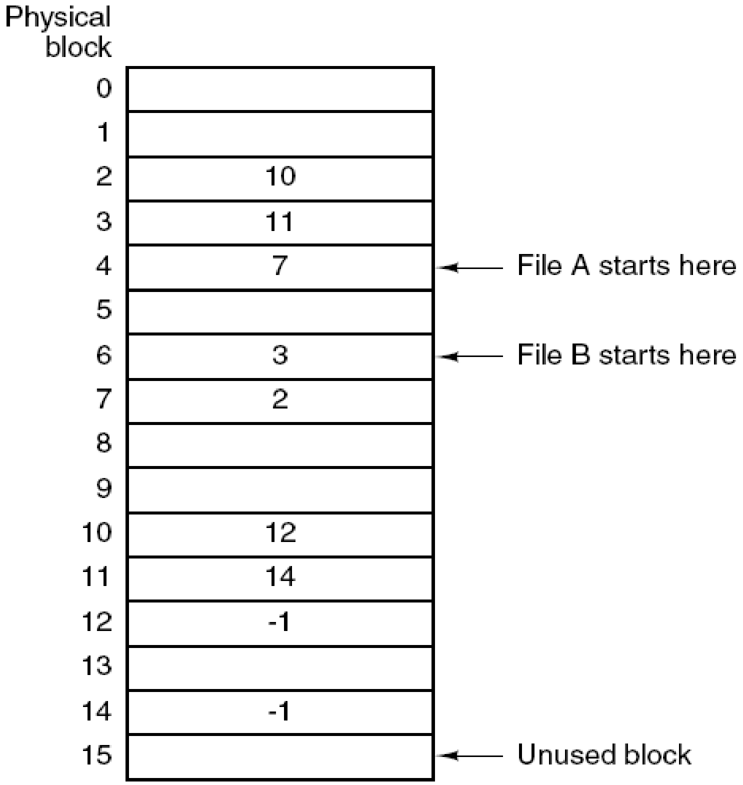
The bad

- Random access is slow. Need to chase pointers to get to a block

# Linked lists using a table in memory

- Put pointers in table in memory
- File Allocation Table (FAT)
- Windows

# The Solution-Linked List Allocation Using a Table in Memory



| Physical block | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | 10 | |
| 3 | 11 | |
| 4 | 7 | File A starts here |
| 5 | | |
| 6 | 3 | File B starts here |
| 7 | 2 | |
| 8 | | |
| 9 | | |
| 10 | 12 | |
| 11 | 14 | |
| 12 | -1 | |
| 13 | | |
| 14 | -1 | |
| 15 | | Unused block |

Figure 4-12. Linked list allocation using a file allocation table in main memory.

# Linked lists using a table in memory

- The bad-table becomes really big
- E.g  200 GB disk with 1 KB blocks needs a 600 MB table
- Growth of the table size is linear with the growth of the disk size

# File System Management and Optimization- the dirty work

- Disk space management
- File System Backups
- File System Consistency
- File System Performance

# Common File Systems

**File Allocation Table (FAT)**

- Wide OS compatibility
- Limit on File size

**New Technology File Systems (NTFS)**

- No constraints on File Size
- File journaling (Enable version recovery)
- Encryption & Authorization
- Lack of compatibility with non-window systems

**Extensible FAT (ExFAT)**

- Optimized for USB and Flash Drives
- Less sophisticated than NTFS
- Better than FAT32
- Unlimited file size
- Broader support

# Common File Systems

**Extended File system (Ext)**

Ext2, Ext3, Ext4

Ext4 has journaling and other sophistications

No native windows or MAC OS support

**Hierarchical File Systems (HFS)**

HFS

HFS+

APFS (optimized for solid state storage devices)

# Which File System to Use?

**System Drive**

Win (NTFS)

Linux (Ext4)

MacOS (HFS+/APFS)

**USB Drive and Memory Cards**

FAT 32 (up to 32 GB)

exFAT (more than 32 GB+)

**Other Drives**

NTFS/ exFAT / HFS+/ APFS (depending on your use and OS preference)