# COP 4610

Operating System Principles

**File System Interface**

1

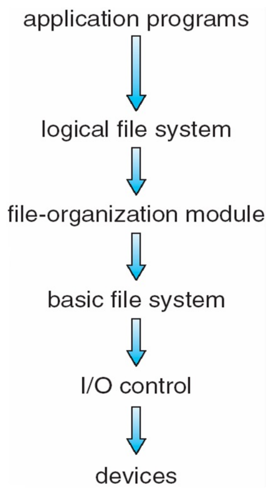
## File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - User interface to storage, mapping logical to physical
  - Efficient and convenient access to disk by allowing data to be stored, located, retrieved easily
- https://en.wikipedia.org/wiki/List_of_file_systems
- File system organized into layers



2

# Layered File System

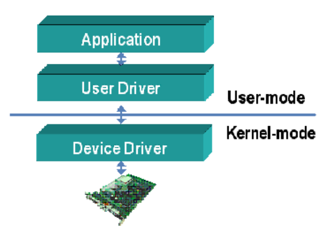





3

# File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive 1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller

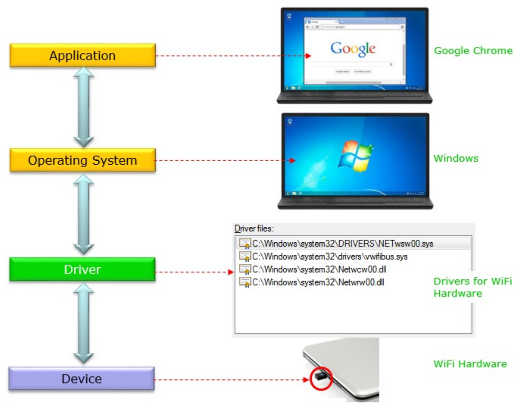





4

# File System Layers

- Device Drivers

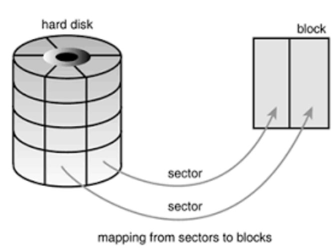



COP 4610 – Operating System Principles 5

5

# File System Layers

- **Basic file system** given command like "retrieve block 123" translates to device driver
  - Also manages memory buffers and caches (allocation, freeing, replacement)
    - – **Buffers** hold data in transit
    - – **Caches** hold frequently used data




COP 4610 – Operating System Principles 6

6

# File System Layers

- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation
  - Sits above the file system
  - “Understands” both sides





# File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location
    - **File control blocks (i-nodes)**
  - Directory management
  - Protection

# File System Layers (Cont.)

- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 40 types, with **extended file system** such as ext2/ext3/ext4 leading; plus distributed file systems, etc.)
  - May newer ones designed for performance, data types, applications, etc.: ZFS, GoogleFS, Oracle ASM, FUSE

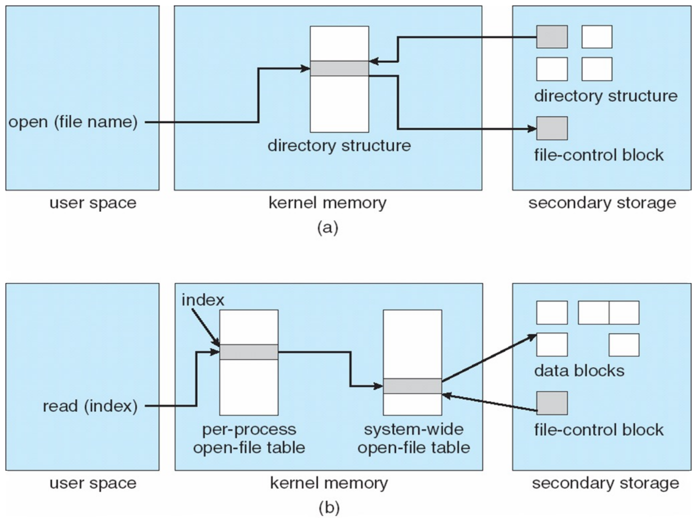




# A Typical File Control Block

| |
|---|
| file permissions |
| file dates (create, access, write) |
| file owner, group, ACL |
| file size |
| file data blocks or pointers to file data blocks |

# In-Memory File System Structures




11

11

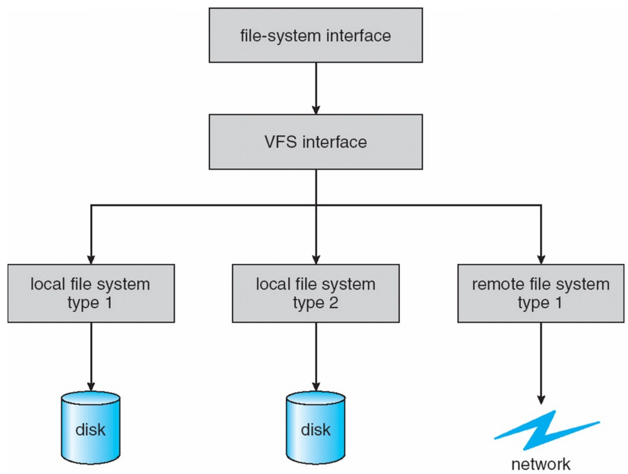
# Virtual File Systems

- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
    - Implements vnodes which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines
- The API is to the VFS interface, rather than any specific type of file system

COP 4610 – Operating System Principles 12

12

# Schematic View of Virtual File System

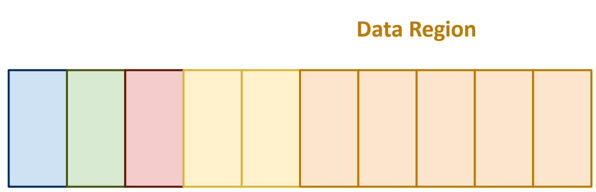



COP 4610 – Operating System Principles 13

13

# File Systems Overview

We need to use **data structures** to organize the data on disk:

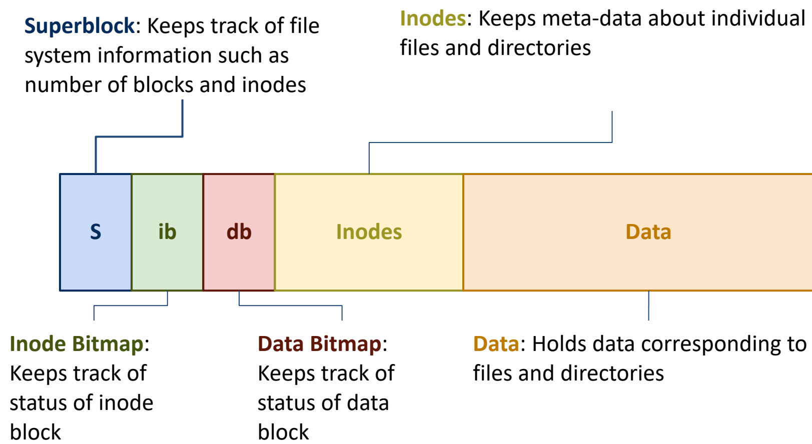



- Divide disk into fixed-sized **blocks** (usually **4**KB)
- Use portion of disk to store **metadata**
- Use another portion to store **allocation structures**
- Use final portion to store **filesystem information**

COP 4610 – Operating System Principles 14

14

# File System Organization






15

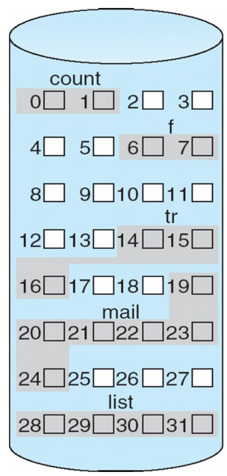
# Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  – Best performance in most cases
  – Simple – only starting location (block #) and length (number of blocks) are required
  – Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**



16

# Contiguous Allocation




directory

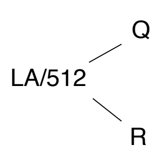
| file | start | length |
|---|---|---|
| count | 0 | 2 |
| tr | 14 | 3 |
| mail | 19 | 6 |
| list | 28 | 4 |
| f | 6 | 2 |





# Contiguous Allocation

- Mapping from logical to physical



# Extent-Based Systems

- Some file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous group of blocks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

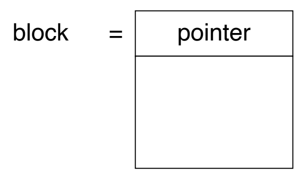


# Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks
- **FAT (File Allocation Table)** variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

# Linked Allocation

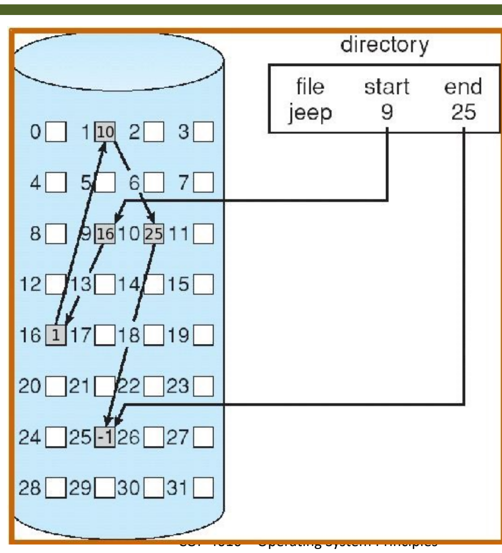
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk






21

# Linked Allocation

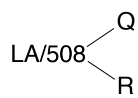





22

# Linked Allocation

- Mapping (Pointer size = 4 bytes)

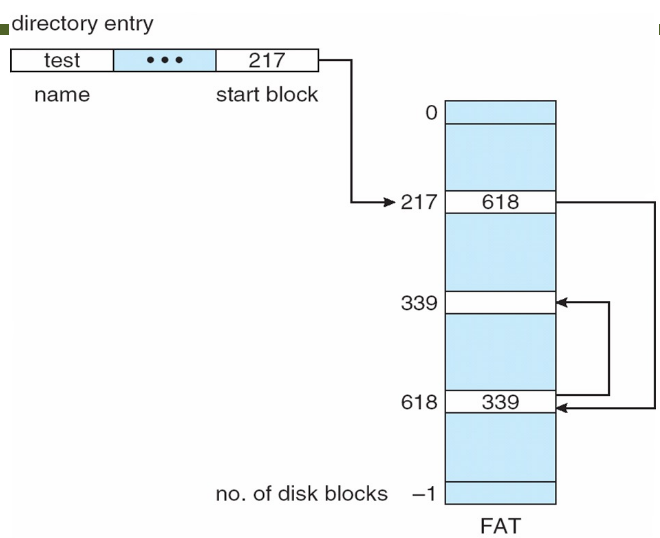



Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 4 (if pointer at beginning of block)



23

# File-Allocation Table

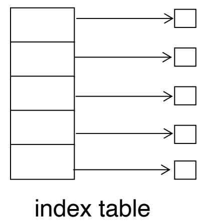





24

# Allocation Methods - Indexed

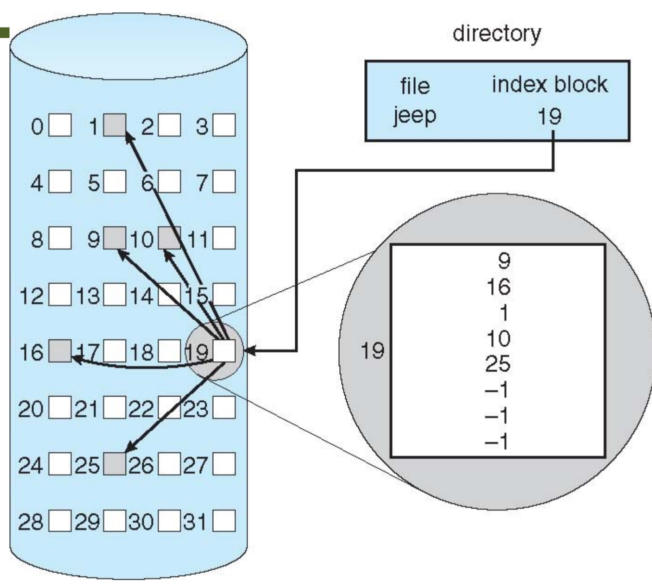
- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks
- Logical view






25

# Example of Indexed Allocation






26

# Indexed Allocation (Cont.)

- Need index table
- Access: index block + data block
- Reliability?
- No external fragmentation
- "Waste" of space? (at least 1 block per file)
- Maximum file size?
  - block size of 512 bytes
  - each pointer = 1 byte
  - size = 256KB
  - larger files: linked list or hierarchical index tables





# Indexed Allocation (Hierarchical)

- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)

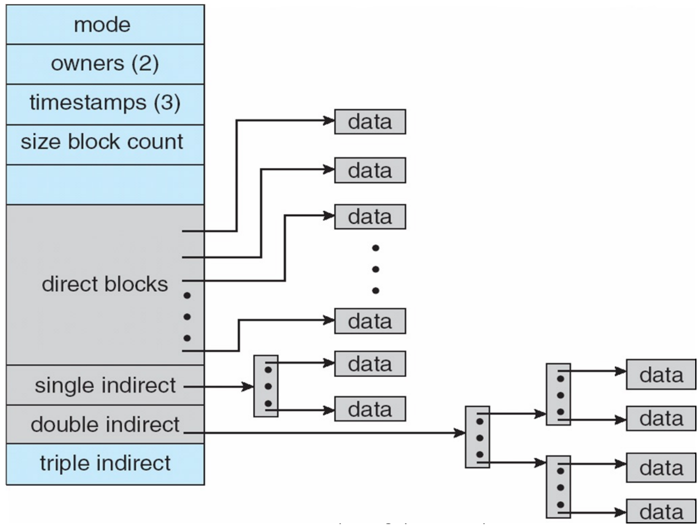
$Q_1$ = displacement into outer-index
$R_1$ is used as follows:

LA / (512 x 512) → $Q_1$, $R_1$

$Q_2$ = displacement into block of index table
$R_2$ displacement into block of file:

$R_1$ / 512 → $Q_2$, $R_2$

## Combined Scheme: UNIX UFS
## (4K bytes per block, 32-bit addresses)

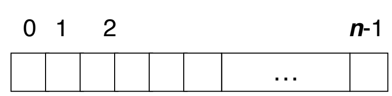







# Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (*n* blocks)




bit[*i*] = { 1 ⇒ block[*i*] free; 0 ⇒ block[*i*] occupied }

**Block number calculation**

(number of bits per word) * (number of 0-value words) + offset of first 1 bit

CPUs have instructions to return offset within word of first "1" bit

# Free-Space Management (Cont.)

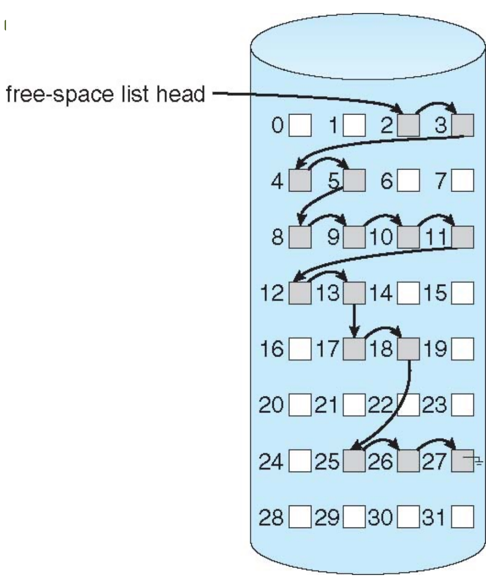
- Bit map requires extra space
  - Example:

    block size = 4KB = $2^{12}$ bytes
    disk size = $2^{40}$ bytes (1 terabyte)
    $n = 2^{40}/2^{12} = 2^{28}$ bits (or 256 MB)
    if clusters of 4 blocks -> 64MB of memory
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)





# Linked Free Space List on Disk




- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)

# Free-Space Management (Cont.)

- Grouping
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts



# Example Part 1

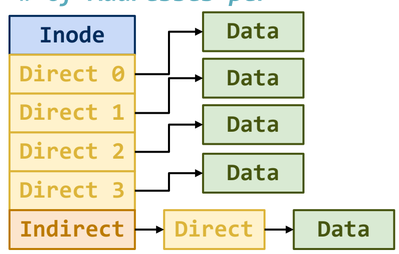
- Consider a **multi-indexed file system** with a **32**-bit block address, **4**KB block size, and an inode structure with **4** direct pointers and one indirect pointer:
  - What is the largest **disk** this file system can use?

Largest Disk = *# of Blocks * Block Size*
= $2^{32}$ * 4KB
= $2^{32} * 2^{12}$
= $2^{44}$
= **16TB**

# Example Part 2

- What is the largest **file** that this file system could store?

largest file = direct blocks + indirect blocks
direct = *# of Direct Blocks * Block Size*
= 4 * 4KB
= $2^2 * 2^{12}$
= $2^{14}$
= **16KB**
indirect = *# of Indirect Blocks * # of Addresses per Block * Block Size*
= 1 * 4KB / 4bytes * 4KB
= $1 * 2^{12} / 2^2 * 2^{12}$
= $2^{22}$
= **4MB**
largest file = **4MB + 16KB**






35

# Example Part 3

- Assuming a disk of **128GB**, how big is the **free block bitmap**?

We need an entry (bit) in our free block bitmap for each block

Free Block Bitmap Size = *Disk Size / Block Size / 8 Bits*
= 128GB / 4KB / 8
= $2^{37} / 2^{12} / 2^3$
= $2^{22}$
= **4MB**

Free Block Bitmap Size = ***$2^{37}$ bytes x block/$2^{12}$ bytes x 1 byte/$2^3$ blocks***



36