Chapter 11. File System Implementation

- File System Structure
- File System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

1. File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers.

Layered File System

- See Figure 11.1.

2. File-System Implementation

- On disk, each file system
  - **Boot control block**, or called boot block in UFS, or called partition boot sector in NTFS
    - First block of a partition
  - **Partition control block**
    - Superblock in UFS
    - Master file block in NTFS
  - Directory structure
    - To organize the files
  - **File control block**
    - Could be stored in directory structure, or
    - Independently
      - Inode in UFS
      - Stored within the MFB in NTFS, which uses a relational database structure
    - See Figure 11.2.

- In memory, to improve performance:
  - Partition control block for each mounted partition
  - Directory structure for recently accessed directories
  - File descriptor tables
  - Open file table
  - Inode table

- See Figure 11.3.

Virtual File Systems

- How to support concurrently multiple types of file system

- **Virtual File Systems (VFS)** 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.
- The API is to the VFS interface, rather than any specific type of file system.

- See Figure 11.4.

Virtual File Systems - continued

- Superblock – in memory
  - Device – device identifier
  - Inode pointers
    - **Mounted inode pointer**
    - Covered inode pointer
  - Block size
  - **Superblock operations**
    - A pointer to a set of superblock routines for this file system
  - File system type
  - File system specific

Virtual File Systems - continued

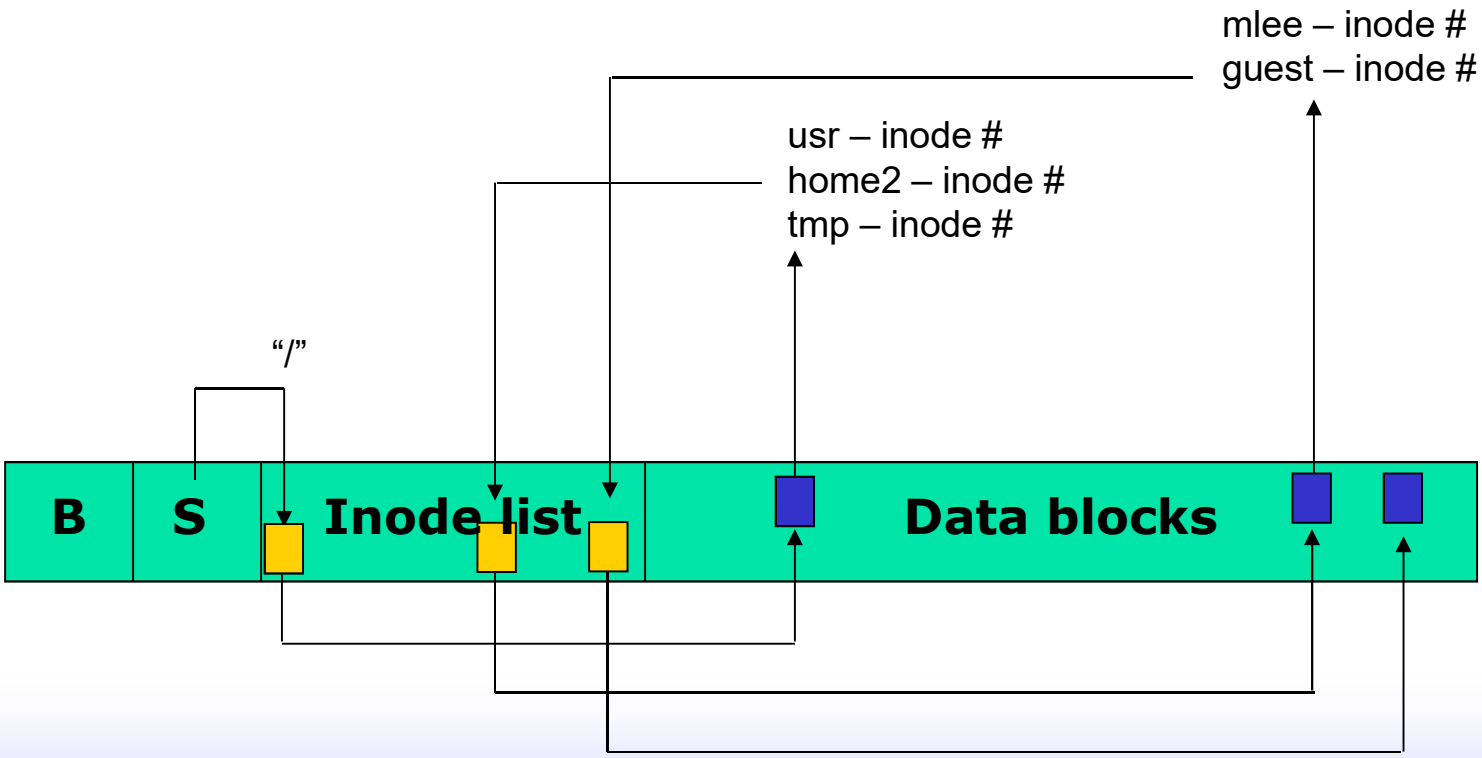
- Inodes for open files and directories – in memory
  - Device – device identifier
  - Inode number
    - The inode number within this file system
  - Mode
    - What it represents
  - User ids
    - The owner identifiers
  - Times
  - Block size
  - Inode operations
  - Count
  - Lock
  - Dirty
  - File system specific information

Virtual File Systems - continued

- Finding a file:
  - From the current working directory – e.g., `$ ls bin`
  - => Inode on the real device for `'.'`
    - File names or directory names with corresponding inodes
  - => Recursive searching ...
  - From the root directory – e.g., `$ ls /usr/src`
  - => Superblock
  - => Device and inode
  - => Inode on the real device for `'/'`
    - File names or directory names with corresponding inodes
  - => Recursive searching…

Virtual File Systems - continued

- Searching a file, e.g., `/home2/guest/tst`



3. Directory Implementation

- Directory contains a list of file names and corresponding additional information how to locate the file.

- **Linear list** of file names with pointer to the data blocks
  - Advantage
    - Simple to program
  - Disadvantage
    - Time-consuming to execute
- **Hash Table** – linear list with hash data structure
  - Advantage
    - Decreases directory search time
  - Disadvantage
    - *Collisions* – situations where two file names hash to the same location => need to use redundant space
    - Fixed size => hard to grow
- **Sorted list** …

4. Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

4.1 Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- See Figure 11.5.

Block size: 512
File: list
File pointer: 1000
Reading 200 bytes
Block to be accessed:
☺ *Which block[s] would be retrieved from the file system?*
Displacement in the first block to be accessed:
☺ *What displacement is the file pointer in the first block to be accessed?*

Contiguous Allocation - continued

- Advantages
  - Simple – Only starting location (block #) and length (number of blocks) are required.
  - Random access
- Disadvantages
  - Wasteful of space (dynamic storage-allocation problem)
    - => external fragmentation
    - Compaction needed
  - Files cannot grow.

Extent-Based Systems

- Modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**.
- An **extent** is a contiguous block of disks. Extents are allocated for file allocation. A file consists of one or more extents.

4.2 Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- See Figure 11.6.

Block size: 512
Size of the link in a block: 4
File: jeep
File pointer: 1020
Reading 100 bytes
Block to be accessed:
☺ *Which block[s] would be retrieved from the file system?*
Displacement in the first block to be accessed:
☺ *What displacement is the file pointer in the first block to be accessed?*

Linked Allocation - continued

- Advantages
  - Simple – need only starting address
  - Free-space management system – no waste of space
- Disadvantages
  - No random access
  - The space required for the pointers
- A solution
  - Use of clusters of multiple blocks

File-Allocation Table (FAT)

- MS-DOS, OS/2
- See Figure 11.7.
  - FAT: not the list of data blocks
  - The memory of index 339 in FAT is -1.
  - -1 in FAT: End of file

Block size: 512
Size of the link in FAT: 4
File: test
File pointer: 1020
Reading 100 bytes
Block to be accessed:
☺ *Which block[s] would be retrieved from the file system?*
Displacement in the first block to be accessed:
☺ *What displacement is the file pointer in the first block to be accessed?*

☺ *The size of FAT?*

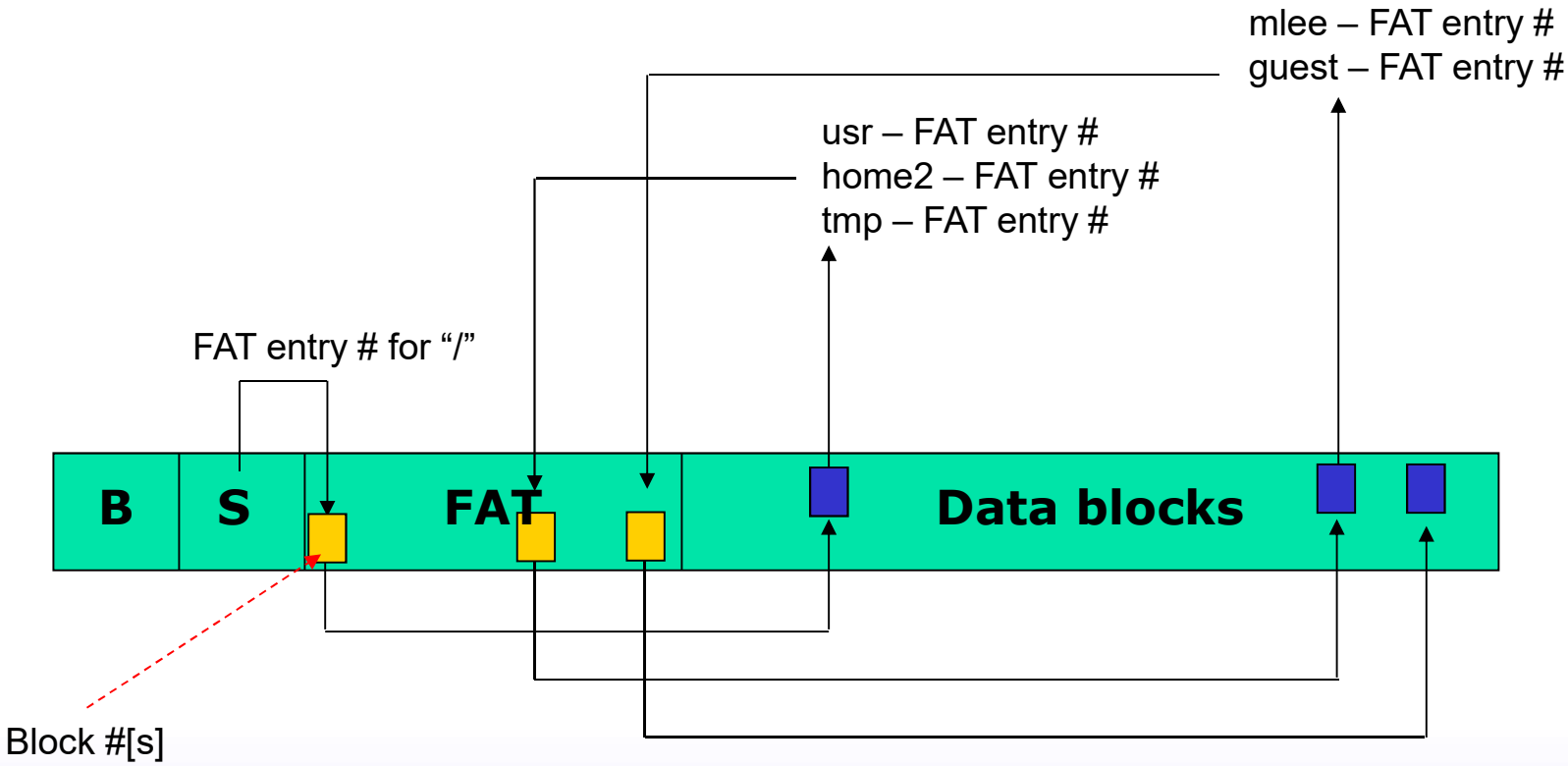
File-Allocation Table - continued

- ☺ *How to allocate a new block to a file?*
  - Another link for unused blocks

- ☺ *Where are directory entry and FAT when the OS is running?*
  - **Directory entry and FAT in main memory to reduce the access time**

- Advantages
  - Actual blocks have no pointer
  - Random access time is improved
- Disadvantages
  - FAT should be in main memory to reduce the access time => difficult to support big file systems

File-Allocation Table - continued



4.3 Indexed Allocation

- Brings all pointers together into the *index block.*
- Logical view
- See Figure 11.8.

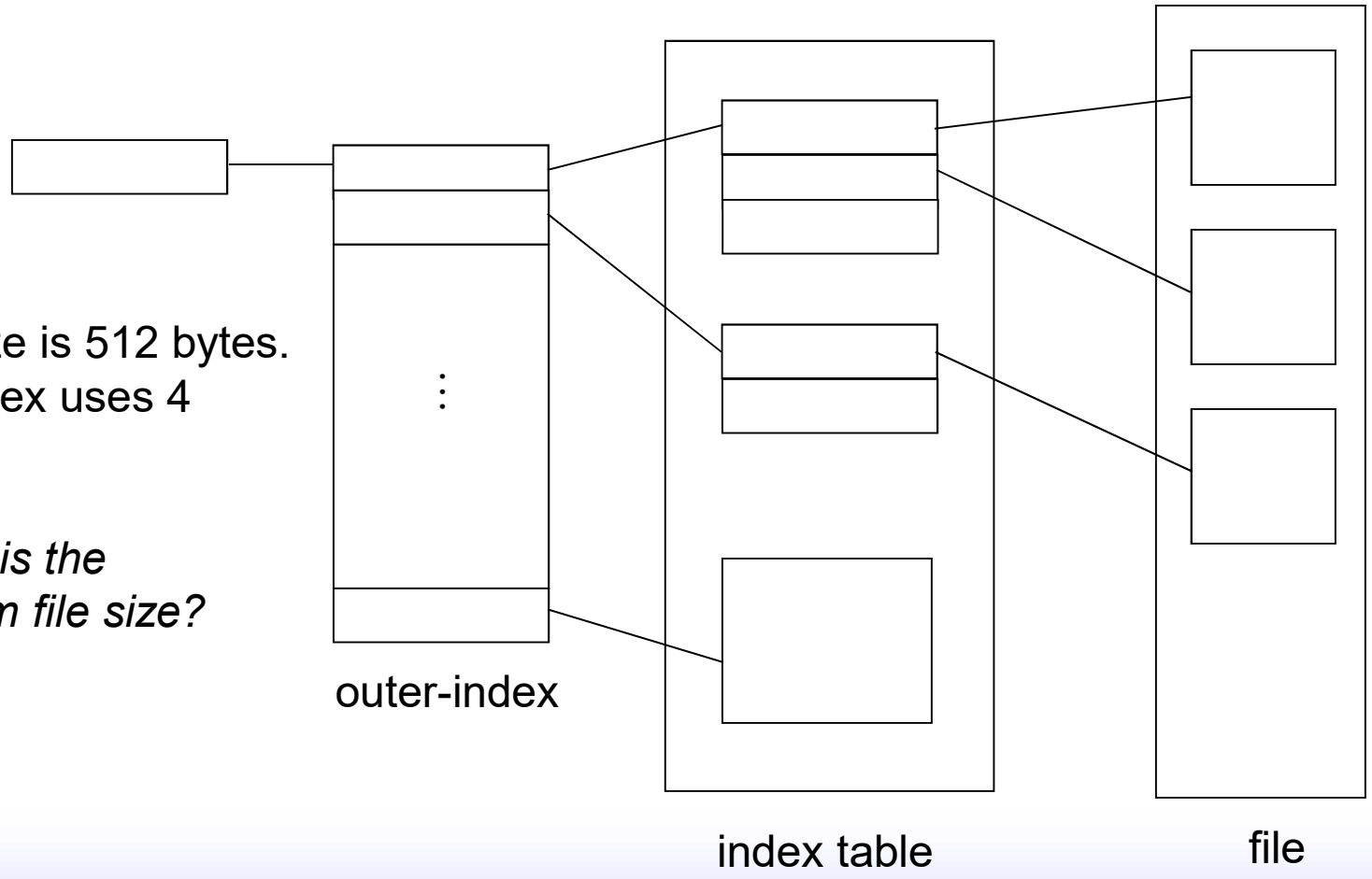
Block size: 512
File: jeep
File pointer: 1020
Reading 100 bytes
Block to be accessed:
☺ *Which block[s] would be retrieved from the file system?*
Displacement in the first block to be accessed:
☺ *What displacement is the file pointer in the first block to be accessed?*

☺ *What is the maximum file size when 4 bytes are used for an index?*

Indexed Allocation - continued

- Advantage
  - Random access
  - No external fragmentation
- Disadvantage
  - Need index table

Two-Level Indexed Allocation



Block size is 512 bytes. Each index uses 4 bytes.

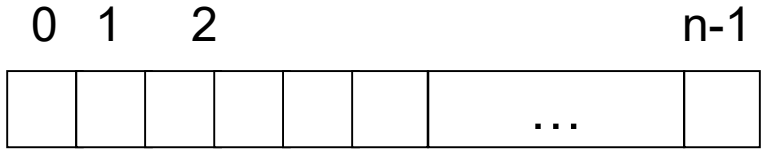
☺ *What is the maximum file size?*

Combined Scheme: UNIX (4K bytes per block)

- See Figure 11.9.
  - 10 direct blocks
  - Indirect block: 170 links
  - ☺ *The total addressable space for a file?*

5. Free-Space Management

- **Bit vector** (*n* blocks)



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

- Example
  - `int bitVector[512];` // int – 4 bytes
  - I would like to see if `block 128` is free.
  - ☺ *Which element of* `bitVector` *should be used?*

Free-Space Management – cont.

- Advantage
  - Easy to get contiguous files
- Disadvantage
  - Bit map requires extra space [in the main memory]
  - Example:

    block size = 2^{12} bytes

    disk size = 2^{30} bytes (1 gigabyte)

    bit map size n = $2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

Free-Space Management – cont.

- **Linked list** (free list)
  - No waste of space
  - Cannot get contiguous space easily
  - Need to access more block to get several free blocks
  - See Figure 11.10.
- **Linked list in FAT**
- **Grouping**
  - A modification of the free-list approach
  - *n* free blocks in the first free block
  - The last free block contains the address of another *n* free blocks, and so on
  - Example: block 2: 3, 4, 5, 8 when four links are allocated to a block
- **Counting**
  - Each entry in the free-space list consists of a block address and a count of contiguous blocks from the block address.
  - Example: block 2: 4, 2; block 4: 8, 2; block 8: 17, 6