CHAPTER 11: FILE SYSTEM IMPLEMENTATION

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

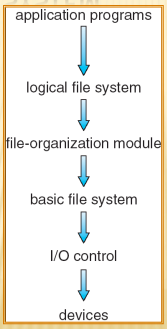
OBJECTIVES

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

FILE-SYSTEM STRUCTURE

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers
- *File control block* – storage structure consisting of information about a file

LAYERED FILE SYSTEM



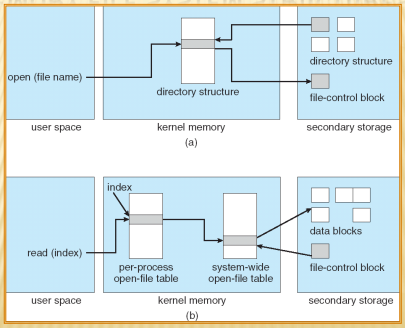
A TYPICAL FILE CONTROL BLOCK

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

- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file.
- Figure 12-3(b) refers to reading a file.

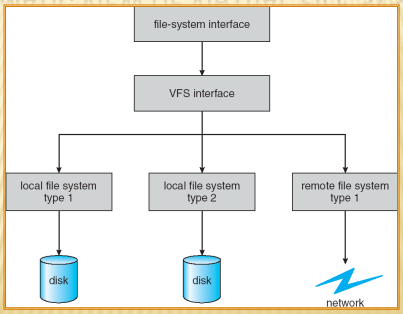
IN-MEMORY FILE SYSTEM STRUCTURES



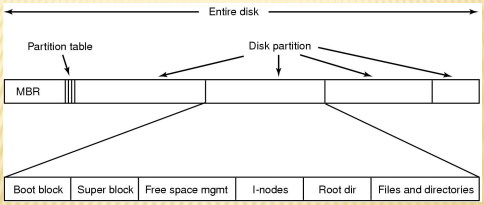
VIRTUAL FILE SYSTEMS

- 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.

SCHEMATIC VIEW OF VIRTUAL FILE SYSTEM



FILE SYSTEM IMPLEMENTATION



A possible file system layout

10

DIRECTORY IMPLEMENTATION

- *Linear list* of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- *Hash Table* – linear list with hash data structure.
  - decreases directory search time
  - *collisions* – situations where two file names hash to the same location
  - fixed size

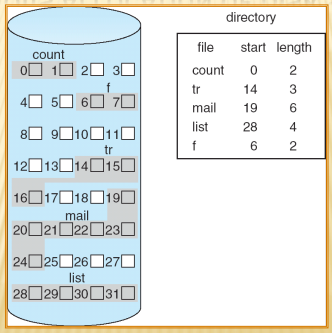
ALLOCATION METHODS

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

CONTIGUOUS ALLOCATION

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

CONTIGUOUS ALLOCATION OF DISK SPACE

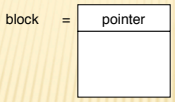


EXTENT-BASED SYSTEMS

- Many newer 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 block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents.

LINKED ALLOCATION

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



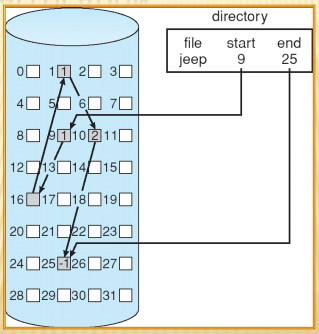
LINKED ALLOCATION (CONT.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

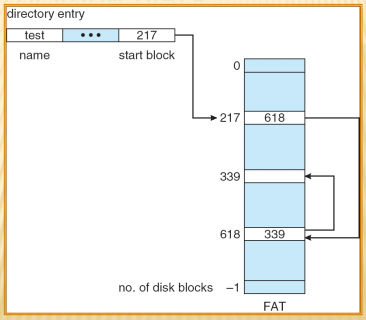
Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 1

File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

LINKED ALLOCATION

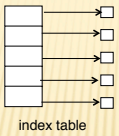


FILE-ALLOCATION TABLE

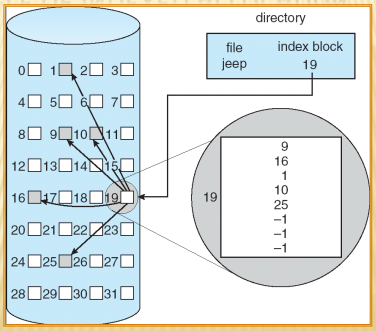


INDEXED ALLOCATION

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



EXAMPLE OF INDEXED ALLOCATION



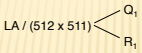
INDEXED ALLOCATION (CONT.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

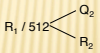
Q = displacement into index table
R = displacement into block

INDEXED ALLOCATION – MAPPING (CONT.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).



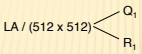
Q_1 = block of index table
R_1 is used as follows:



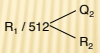
Q_2 = displacement into block of index table
R_2 displacement into block of file:

INDEXED ALLOCATION – MAPPING (CONT.)

- Two-level index (maximum file size is 512^3)

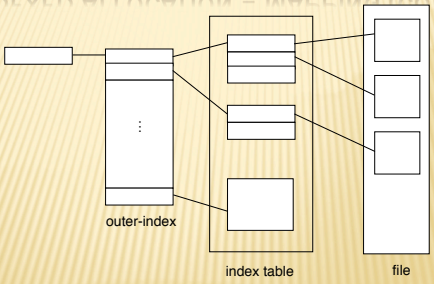


Q_1 = displacement into outer-index
R_1 is used as follows:

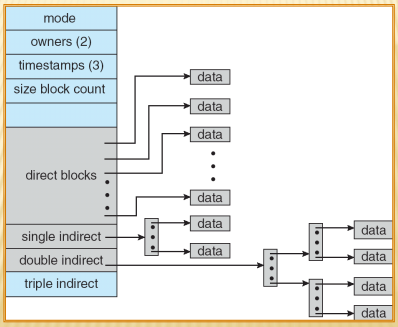


Q_2 = displacement into block of index table
R_2 displacement into block of file:

INDEXED ALLOCATION – MAPPING (CONT.)

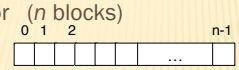


COMBINED SCHEME: UNIX (4K BYTES PER BLOCK)



FREE-SPACE MANAGEMENT

- Bit vector (*n* blocks)



$$bit[i] = \begin{cases} 0 \Rightarrow block[i] \text{ free} \\ 1 \Rightarrow block[i] \text{ occupied} \end{cases}$$

Block number calculation

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

FREE-SPACE MANAGEMENT (CONT.)

- Bit map requires extra space
  - Example:
    block size = 2^{12} bytes
    disk size = 2^{30} bytes (1 gigabyte)
    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting

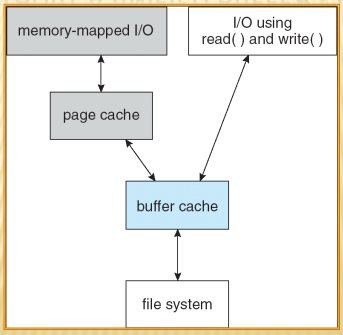
EFFICIENCY AND PERFORMANCE

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

PAGE CACHE

- A *page cache* caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

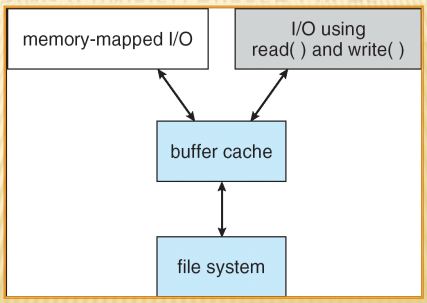
I/O WITHOUT A UNIFIED BUFFER CACHE



UNIFIED BUFFER CACHE

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O

I/O USING A UNIFIED BUFFER CACHE



RECOVERY

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by *restoring* data from backup

LOG STRUCTURED FILE SYSTEMS

- *Log structured* (or journaling) file systems record each update to the file system as a *transaction*
- All transactions are written to a *log*
  - A transaction is considered *committed* once it is written to the log
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed

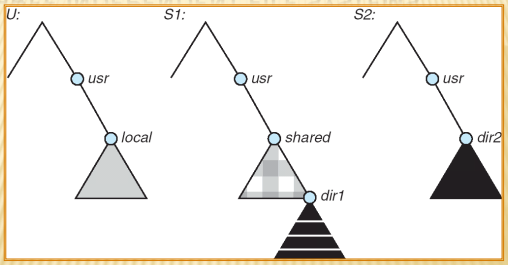
THE SUN NETWORK FILE SYSTEM (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet)

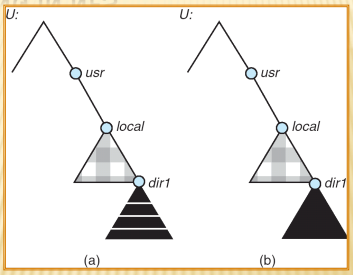
NFS (CONT.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

THREE INDEPENDENT FILE SYSTEMS



MOUNTING IN NFS

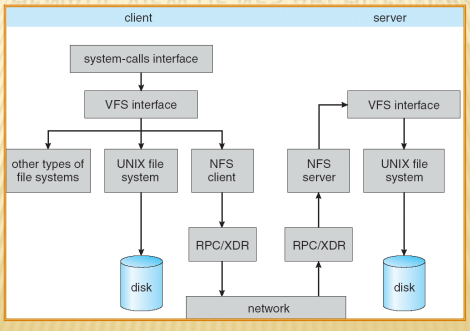


Mounts Cascading mounts

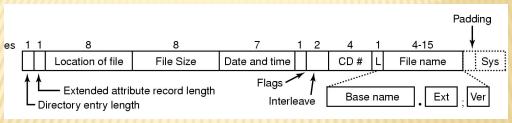
THREE MAJOR LAYERS OF NFS ARCHITECTURE

- UNIX file-system interface (based on the *open, read, write*, and *close* calls, and *file descriptors*)
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
  - Implements the NFS protocol

SCHEMATIC VIEW OF NFS ARCHITECTURE

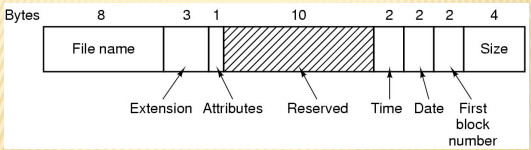


EXAMPLE FILE SYSTEMS
CD-ROM FILE SYSTEMS



The ISO 9660 directory entry

THE MS-DOS FILE SYSTEM (1)



The MS-DOS directory entry

43

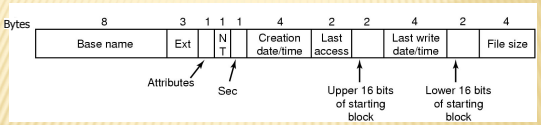
THE MS-DOS FILE SYSTEM (2)

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

- Maximum partition for different block sizes
- The empty boxes represent forbidden combinations

44

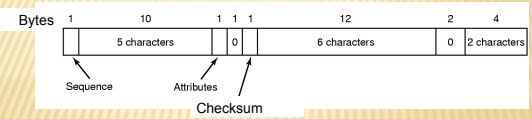
THE WINDOWS 98 FILE SYSTEM (1)



The extended MOS-DOS directory entry used in Windows 98

45

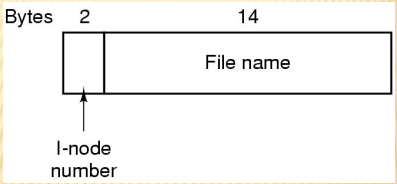
THE WINDOWS 98 FILE SYSTEM (2)



An entry for (part of) a long file name in Windows 98

46

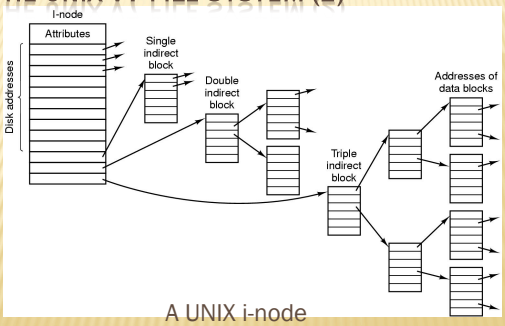
THE UNIX V7 FILE SYSTEM (1)



A UNIX V7 directory entry

47

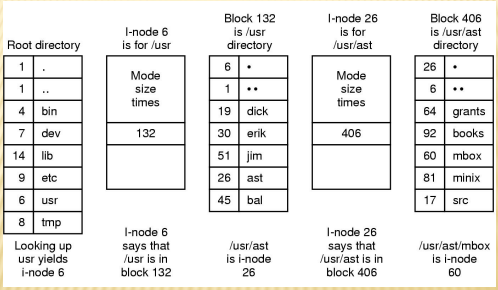
THE UNIX V7 FILE SYSTEM (2)



A UNIX i-node

48

THE UNIX V7 FILE SYSTEM (3)



The steps in looking up */usr/ast/mbox*

49

END OF CHAPTER 11