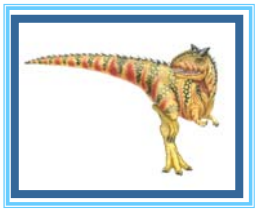
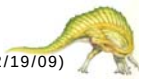
Chapter 11: File System Implementation



Chapter 11: File System Implementation

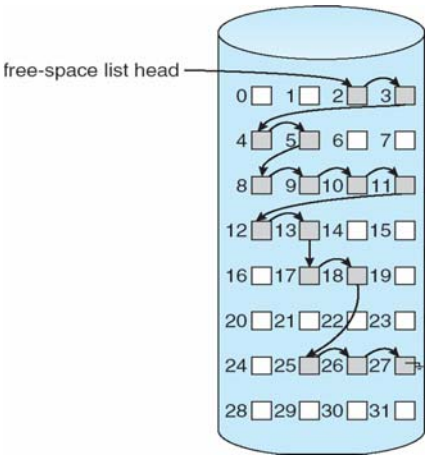
- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS
- Example: WAFL File System

(slides selected/reordered/fixed by R. Doemer, 02/19/09)



Free-Space Management

- Linked free space list on disk



(slide updated by R. Doemer, 02/19/09)

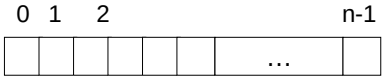
Free-Space Management (Cont.)

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Need to protect:
  - Pointer to free list
- Extensions
  - Grouping
    - Multiple block pointers in one block
  - Counting
    - List of blocks, each block represents *n* contiguous blocks
- Alternatives
  - Store “free file” in FAT or indexed allocation scheme
  - Bitmap scheme (next page)

(slide fixed by R. Doemer, 02/19/09)

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 (4kB)
    disk size = 2^{40} bytes (1 terabyte)
    $n = 2^{40}/2^{12} = 2^{28}$ bits (32 megabyte)
- Easy to get contiguous files
- Bit map must be kept on disk
  - Copy in memory and disk may differ
  - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
  - Solution:
    - Set bit[*i*] = 1 in disk
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

(slide combined/updated by R. Doemer, 02/19/09)

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

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

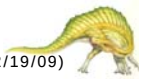
Log Structured File Systems

- **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

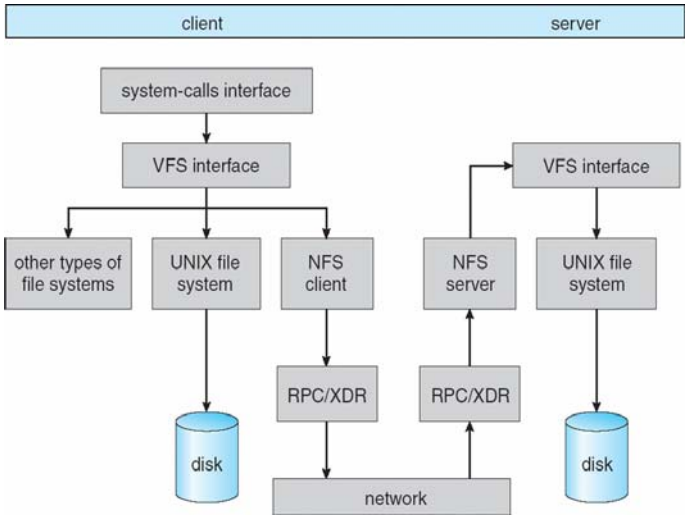
The Sun Network File System (NFS)

- An implementation and a specification of a software file 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, e.g. over Ethernet)
- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures
  - NFS specification is independent of these media
  - Independence is achieved through the use of remote procedure call (RPC) primitives used between two implementation-independent file system interfaces

(slides combined/simplified by R. Doemer, 02/19/09)



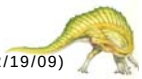
Schematic View of NFS Architecture



NFS Architecture Layers

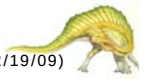
- UNIX file-system interface
  - based on **open, read, write**, and **close** calls,
  - and **file descriptors**
- *Virtual File System* (VFS) layer
  - Activates file-system-specific operations to handle requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- NFS service layer
  - Bottom layer of the architecture
  - Implements the NFS protocol

(slides combined/simplified by R. Doemer, 02/19/09)



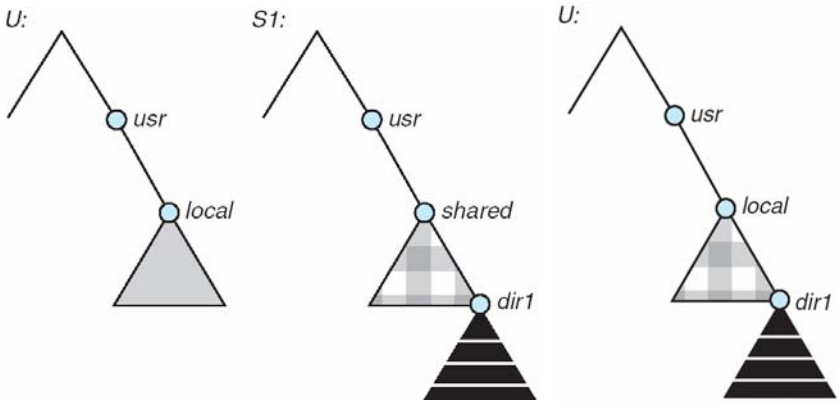
NFS (Cont.)

- Interconnected workstations are viewed as a set of independent machines with independent file systems
  - A remote directory is mounted over a local file system directory
    - The mounted directory looks like an integral sub-tree of the local file system
    - Unless empty, it replaces the subtree descending from the local directory
  - Specification of the remote directory for the mount operation is nontransparent
    - Host and full name of the remote directory have to be provided
    - Files in the remote directory can then be accessed in a transparent manner



(slides combined/simplified by R. Doemer, 02/19/09)

NSF Mounting

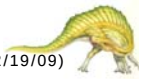


Local file system Remote file system Mounted as NFS

(slides combined/simplified by R. Doemer, 02/19/09)

NFS Mount Protocol

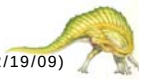
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services
- NFS mount protocol establishes an initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle — a key for further accesses
- The mount operation changes only the user's view and does not affect the server side



(slides combined/simplified by R. Doemer, 02/19/09)

NFS Protocol

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
  - searching for a file within a directory
  - reading a set of directory entries
  - manipulating links and directories
  - accessing file attributes
  - reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments
- Modified data must be committed to the server's disk before results are returned to the client
- The NFS protocol does not provide concurrency-control mechanisms



(slides combined/simplified by R. Doemer, 02/19/09)

End of Chapter 11