CSE 30341

Operating System Principles

File System Interface

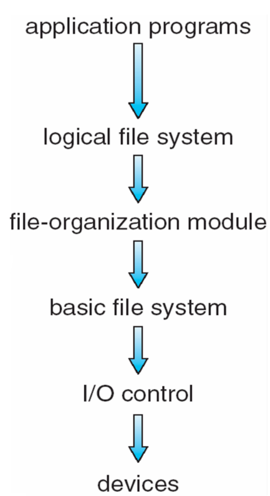
Objectives

- 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)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- **File control block** (FCB) – storage structure consisting of information about a file ("**i-node**")
- File system organized into layers

Layered File System



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

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

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**
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
  - Logical layers can be implemented by any coding method according to OS designer

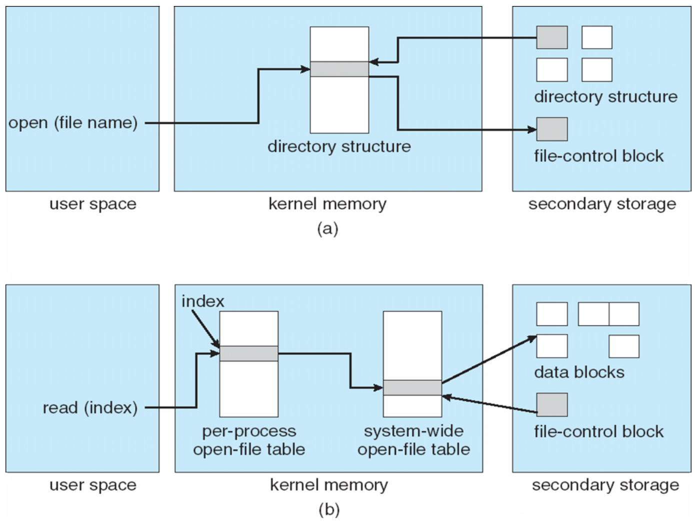
File System Layers (Cont.)

- 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.)
  - New ones still arriving – 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



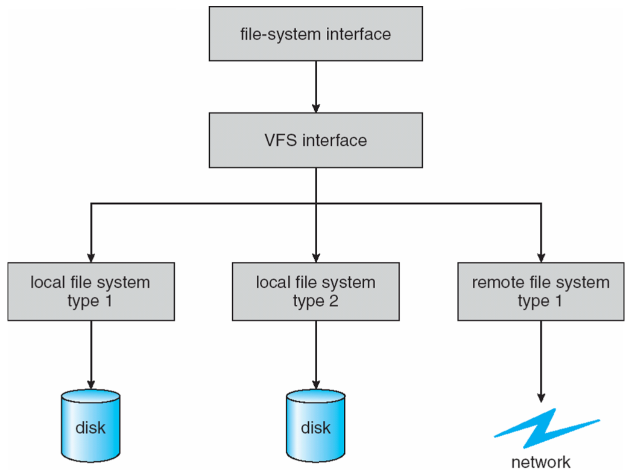
11

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

CSE 30341 – Operating System Principles 12

Schematic View of Virtual File System



Virtual File System Implementation

- For example, Linux has four object types:
  - i-node, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
  - Every object has a pointer to a function table
    - Function table has addresses of routines to implement that function on that object

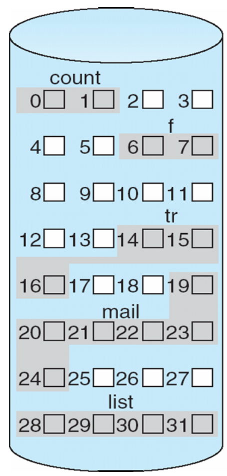
Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **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 entries or use chained-overflow method

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**

Contiguous Allocation

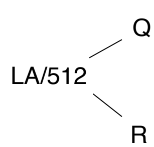


directory

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

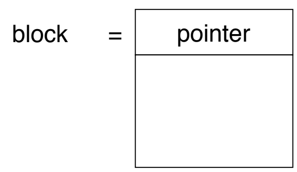
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous group of blocks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Allocation Methods - Linked

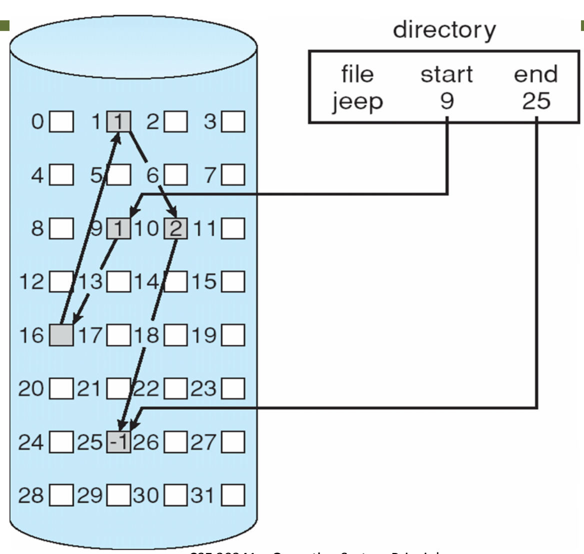
- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

Linked Allocation

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

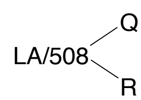


Linked Allocation



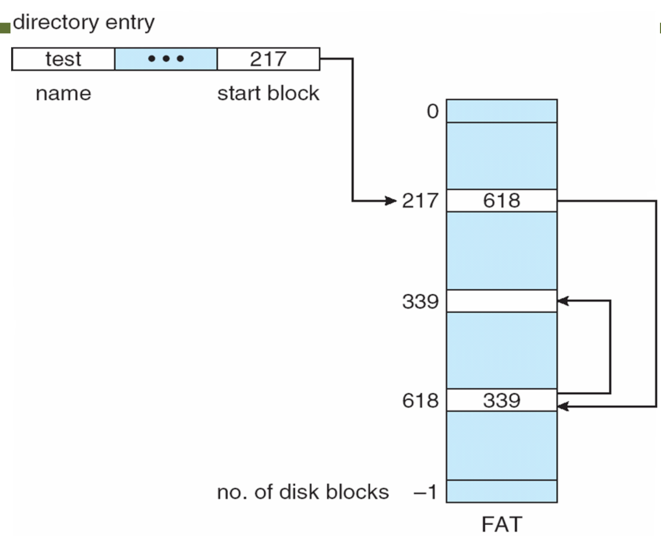
Linked Allocation

- Mapping (Pointer size = 4 bytes)



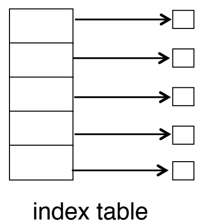
Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 4 (if pointer at beginning of block)

File-Allocation Table

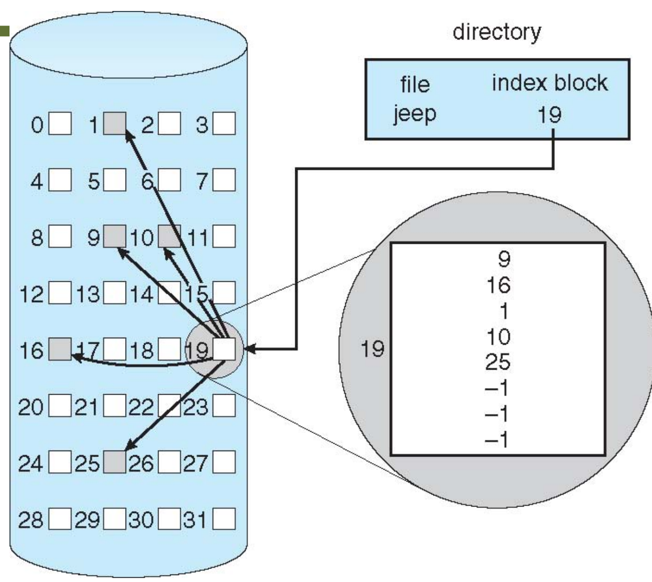


Allocation Methods - Indexed

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



Example of Indexed Allocation



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

- Mapping from logical to physical in a file of unbounded length (block size of 512 bytes; pointer size = 1 byte)

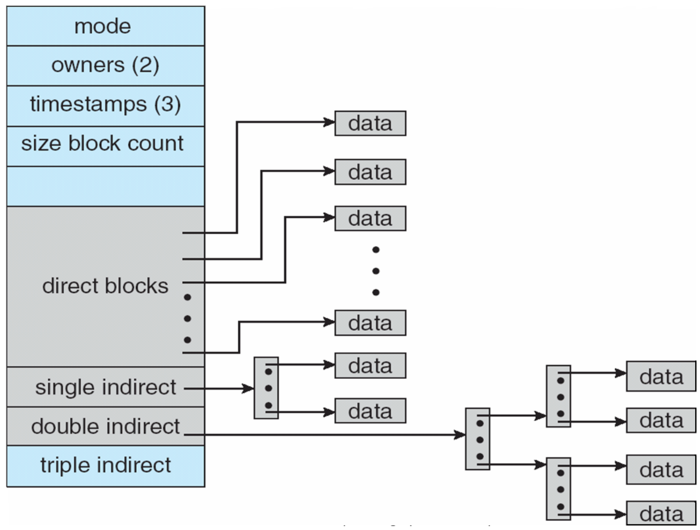
Q_1 = block of index table
R_1 is used as follows:

LA / (512 x 511) → Q_1, R_1

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

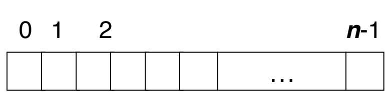
R_1 / 512 → Q_2, R_2

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)



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

Block number calculation

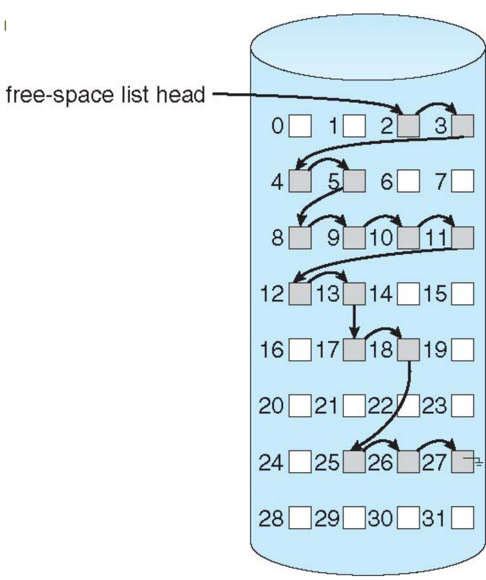
(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

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

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.)

- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A remote directory is mounted over a local file system directory
    - The mounted directory looks like an integral subtree of the local file system
    - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
    - Files in the remote directory can then be accessed in a transparent manner
  - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

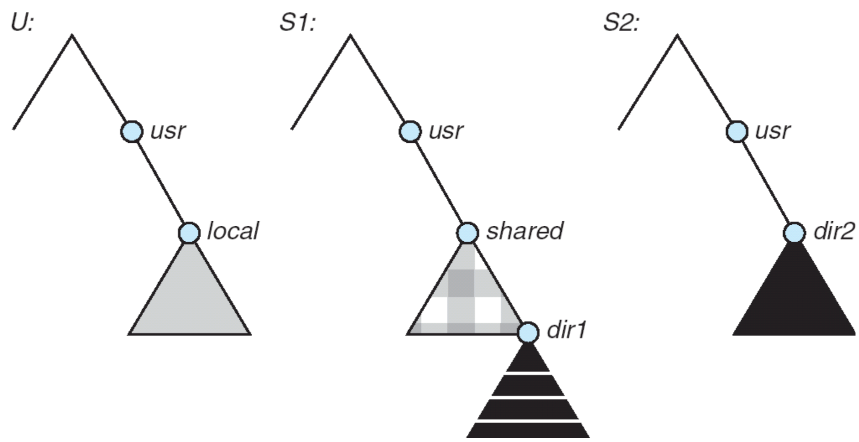
NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media

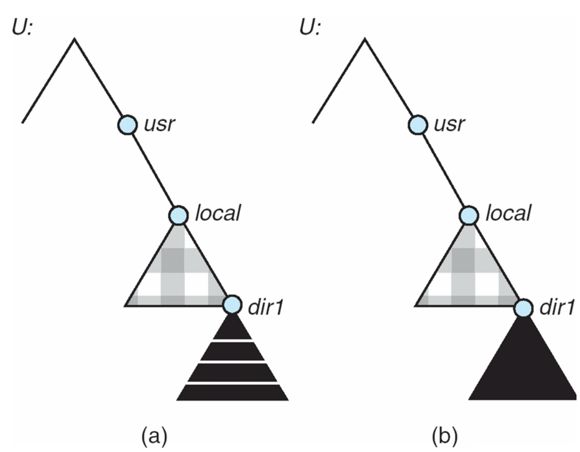
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces

- 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

NFS Mount Protocol

- Establishes 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
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

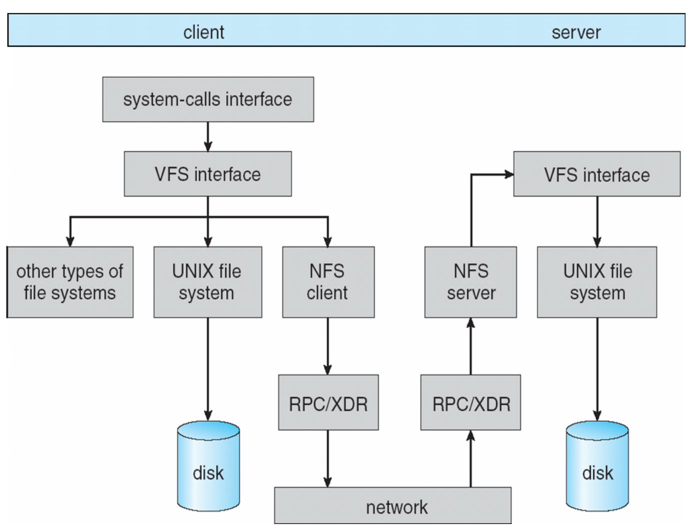
NFS Protocol

- 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 (NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms

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



NFS Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk