Chương 7: File system

Tìm hiểu cách thức tổ chức thông tin trên đĩa. File & quản lý File

Nội dung

- Part 1: File Interface
  - Khái niệm File
  - Các phương pháp truy nhập - Access Methods
  - Cấu trúc thư mục - Directory Structure
  - Chia sẻ file - File Sharing
  - Protection
- Part 2: Implemention File System

1.1. Khái niệm(1)

- File là một tập hợp của các thông tin liên quan, được ghi trên bộ nhớ thứ cấp (lưu trữ lâu dài) và được đặt tên.
- Từ góc nhìn của người sử dụng, file là đơn vị bộ nhớ logic nhỏ nhất. Các file được ánh xạ bởi HĐH vào các thiết bị nhớ vật lý.
- Kiểu File:
  - Data
  - số - numeric
  - ký tự - character
  - nhị phân - binary
  - Program
- Nói chung, file là một chuỗi các bit, byte, dòng hoặc bản ghi

1.1. Khái niệm(2)

- Cấu trúc File:
  - Có thể không cấu trúc - chuỗi các words, bytes
  - Cấu trúc bản ghi đơn giản
    - các dòng (lines)
    - độ dài cố định
    - độ dài thay đổi
  - Các cấu trúc phức tạp
    - văn bản có định dạng - Formatted document
    - file nạp có thể tái định vị - Relocatable load file
  - Ai quyết định cấu trúc file?
    - HĐH
    - Chương trình

1.1. Khái niệm(3)

- Thuộc tính File
  - **Name** – chỉ là thông tin ở dạng người đọc được.
  - **Type** – cần thiết cho các HĐH hỗ trợ nhiều kiểu file.
  - **Location** – con trỏ tới vị trí file trên thiết bị.
  - **Size** – kích thước hiện tại của file.
  - **Protection** – kiểm soát ai có thể đọc, ghi, thực hiện file.
  - **Time**, **date**, **user identification** – dữ liệu dùng cho protection, security, và theo dõi sử dụng.
  - Thông tin về file được lưu trong cấu trúc thư mục, cũng được lưu trên đĩa.

1.1. Khái niệm(4)

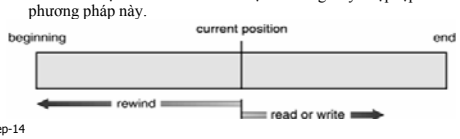
- Các Thao tác với File
  - Tạo file
  - Ghi file
  - Đọc file
  - Định vị trong file – file seek
  - Xóa file
  - Cắt bớt file (truncate)
  - Open(*Fi*) – tìm chỉ mục Fi trong cấu trúc thư mục trên đĩa rồi chuyển nội dung của chỉ mục vào bộ nhớ.
  - Close (*Fi*) – chuyển nội dung của chỉ mục Fi trong bộ nhớ ra cấu trúc thư mục trên đĩa.

1.1. Khái niệm(5): File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

1.2. Các phương pháp truy nhập(1)

- Truy nhập tuần tự - Sequential Access
  - Truy nhập tuần tự qua các bản ghi từ đầu tệp đến cuối tệp
    - *read next*
    - *write next*
    - *reset*
    - no *read after last write*
  - Một số HĐH cho phép nhảy tới hoặc lui *n* bản ghi.
  - Các trình soạn thảo và trình biên dịch thường truy nhập tệp theo phương pháp này.



1.2. Các phương pháp truy nhập(2)

- Truy nhập trực tiếp - Direct Access
  - Tệp được tạo bởi các bản ghi có kích thước cố định
  - Có thể truy nhập các bản ghi tại vị trí bất kỳ trong tệp mà không cần theo thứ tự.
  - Các CSDL thường được tổ chức theo phương pháp này
  - Sử dụng các phương thức:
    - *read n*
    - *write n*
    - *position to n*
    - *read next*
    - *write next*
    - *rewrite n*
  - *n* = số hiệu bản ghi cần truy nhập, có thể bắt đầu từ 0 hoặc 1 tùy thuộc HĐH

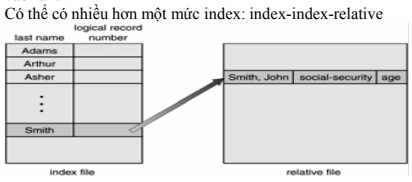
1.2. Các phương pháp truy nhập(3)

- Minh hoạ Truy nhập trực tiếp
  - cp - current position: biến xác định vị trí hiện tại

sequential access	implementation for direct access
reset	*cp = 0;*
read next	*read cp;* *cp = cp+1;*
write next	*write cp;* *cp = cp+1;*

1.2. Các phương pháp truy nhập(4)

- Truy nhập index-relative
  - File index chứa các con trỏ tới các bản ghi trong File relative
  - Để truy nhập các bản ghi trong File relative, trước tiên tìm index, tiếp theo dùng con trỏ để truy nhập trực tiếp File relative để tìm bản ghi.
  - Hữu dụng khi tìm kiếm trong các File lớn vì số lần thực hiện vào-ra ít
  - Có thể có nhiều hơn một mức index: index-index-relative

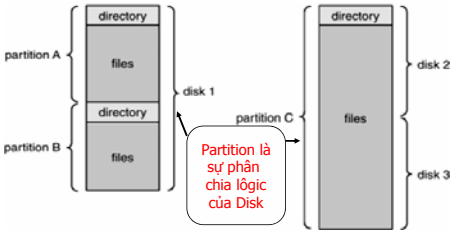


1.3. Cấu trúc thư mục(1)

- Để quản lý số lượng lớn các tệp trên đĩa, tổ chức chúng trong 2 bước:
  - Đầu tiên, chia đĩa thành một hay nhiều *partition* (minidisk-IBM, volume-PC & Macintosh)
    - Partition - cấu trúc mức thấp, để chứa các tệp và thư mục
    - Một số HĐH cho phép patition lớn hơn đĩa
  - Tiếp theo, mỗi partition có một **device directory** ghi thông tin về tất cả các File trên Partition đó: tên file, vị trí, kích thước, kiểu file…

1.3. Cấu trúc thư mục(2)

- Một tổ chức hệ thống file cơ bản



1.3. Cấu trúc thư mục(3)

- Các thao tác trên một thư mục
  - Tìm kiếm 1 tệp
  - Tạo 1 tệp
  - Xóa 1 tệp
  - Liệt kê danh sách tệp trong thư mục
  - Đổi tên 1 tệp
  - Truy nhập toàn bộ hệ thống file

1.3. Cấu trúc thư mục(4)

- Tổ chức logic thư mục để sử dụng
  - **Hiệu quả (Efficiency)** – định vị file nhanh chóng.
  - **Đặt tên (Naming)** – thuận tiện cho người sử dụng.
    - Nhiều file có thể có cùng tên.
    - 1 File có thể có nhiều tên.
  - **Gom nhóm (Grouping)** – nhóm logic các file theo **thuộc tính**, (vd: all Java programs, all games, …)

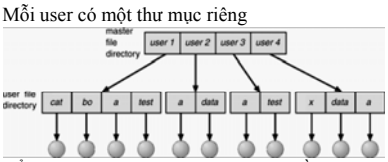
1.3.1. Single-Level Directory

- Một thư mục chứa tất cả các tệp.

- Ưu:
  - dễ hiểu, dễ quản lý
  - kích thước nhỏ
- Nhược:
  - vấn đề đặt tên: mỗi tệp phải có tên duy nhất
  - vấn đề gom nhóm: không thể

1.3.2. Two-Level Directory

- Mỗi user có một thư mục riêng

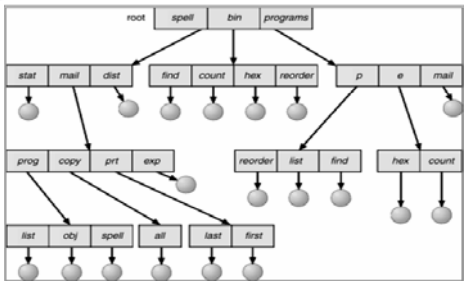


- Để truy nhập 1 tệp ở thư mục khác, cần có đường dẫn đầy đủ
- Có thể có các tệp trùng tên cho các user khác nhau
- Tìm kiếm hiệu quả hơn
- Không có khả năng gom nhóm

1.3.3. Tree-Structured Directories(1)

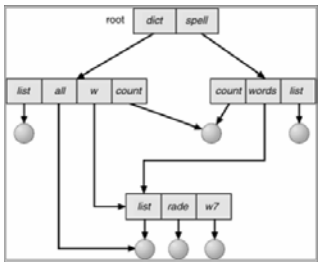
- Có 1 thư mục gốc (root)
- Mỗi tệp có 1 đường dẫn duy nhất:
  - Đầy đủ, vd: **C:\Windows\php.ini**
  - Quan hệ (với thư mục hiện tại), vd: **.\System32\test.dll**
- Mỗi thư mục chứa các tệp và/hoặc các thư mục con
- Tìm kiếm hiệu quả
- Thuận tiện trong đặt tên
- Có khả năng gom nhóm

1.3.3. Tree-Structured Directories(2)



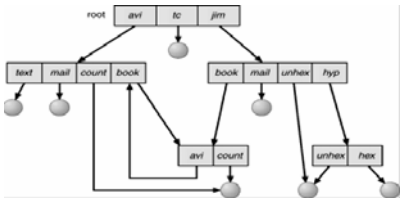
1.3.4. Acyclic-Graph Directories

- Có sự chia sẻ (not copy!) các thư mục con và các tệp, thuận tiện khi nhiều user làm việc trong 1 dự án.



1.3.5. General Graph Directory

- Khi một liên kết được thêm vào cấu trúc, cần đảm bảo không tạo thành chu trình → sử dụng giải thuật tìm kiếm chu trình trong đồ thị, nhưng là việc "nặng nhọc" vì đồ thị trên đĩa, không phải trong bộ nhớ trong.



1.4. File Sharing

- Yêu cầu: phải chia sẻ các file trên các hệ thống đa người dùng (multi-user systems).
- Chia sẻ file có thể được thực hiện thông qua một lược đồ *protection*.
- Trên các hệ thống phân tán (distributed systems), các file có thể được chia sẻ qua mạng (network).
- Network File System (NFS) là một phương thức chia sẻ file phân tán (distributed file-sharing method) phổ biến.
  - sử dụng trong mô hình client-server
  - các user ID phải phù hợp cả với client và server để xác nhận quyền truy nhập file trên server.

1.5. Protection(1)

- Người tạo/sở hữu file cần có khả năng giám sát:
  - thao tác nào đã được thực hiện
  - bởi user nào?
- Các loại truy nhập
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

1.5. Protection(2): Access Lists and Groups

- Chế độ truy nhập: read, write, execute
- Ba lớp người sử dụng:

			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

- Yêu cầu người quản lý tạo một group G (có tên duy nhất), rồi thêm các user vào group.
- Đối với các file (vd *game*) hoặc subdirectory, xác định sự truy nhập tương tự như trên.

Gắn group cho file:

chgrp G game

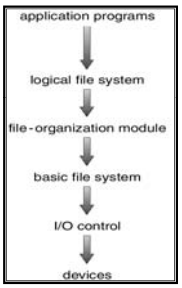
Part 2 : Implemention File System

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

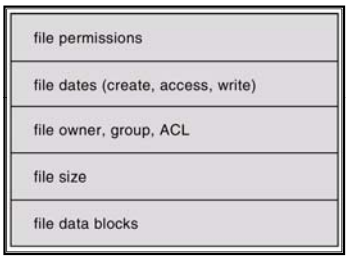
2.1. File-System Structure & Implementation

- 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.

2.1.1. Layered File System



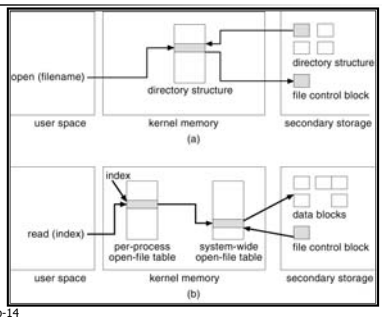
2.1.2. A Typical File Control Block



2.1.3. In-Memory File System Structures

- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 2.1.4(a) refers to opening a file.
- Figure 2.1.4(b) refers to reading a file.

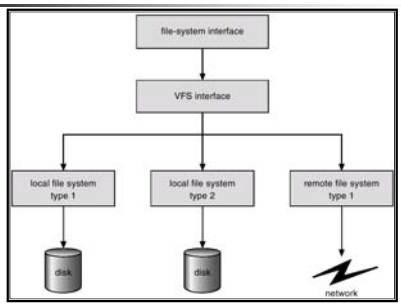
2.1.4. In-Memory File System Structures



2.1.5. Virtual File Systems(1)

- 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.

2.1.5. Schematic View of Virtual File System(2)



2.2. Directory Implementation

- Linear list(danh sách tuyến tính) of file names with **pointer** to the data blocks.
  - Simple to program
  - Time-consuming to execute(thời gian thực thi không tốt)
- Hash Table – linear list with hash data structure.
  - Decreases directory search time
  - *Collisions* – situations(vị trí xung đột) where two file names hash to the same location
  - Fixed size

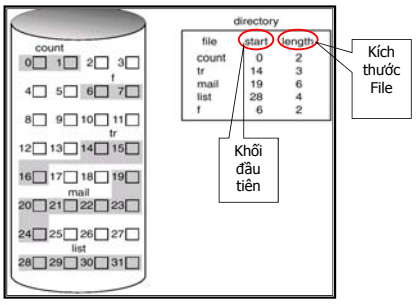
2.3. Allocation Methods

- An **allocation method** refers to **how disk blocks are allocated for files**:
  - Contiguous allocation(Định vị liên tục)
  - Linked allocation(Định vị liên kết)
  - Indexed allocation(Định vị chỉ số)

2.3.1. Contiguous Allocation(1)

- Each file occupies(chiếm giữ) a set of contiguous blocks on the disk(tập liên tiếp các khối).
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access.
- Wasteful of space (dynamic storage-allocation problem)-lãng phí không gian nhớ
  - Có thể còn rất nhiều vùng nhớ trống rời rạc nhưng mỗi vùng đều không đủ lưu trữ trọn vẹn một file
- Files **cannot grow**.

2.3.1. Contiguous Allocation of Disk Space(2)



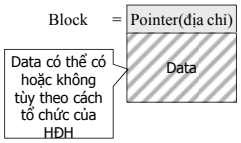
2.3.1. Extent-Based Systems(3)

- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme.
- Extent-based file systems allocate disk blocks in **extents**.-Hệ thống file định vị trên disk theo miền(khu vực - extent)
- An **extent** is **a contiguous block** of disks. Extents are allocated for file allocation. **A file consists of one or more extents**.

⇒ File gồm một tập các miền rời rạc, trên mỗi miền lại được tổ chức thành từng khối liên tục

2.3.2. Linked Allocation(1)

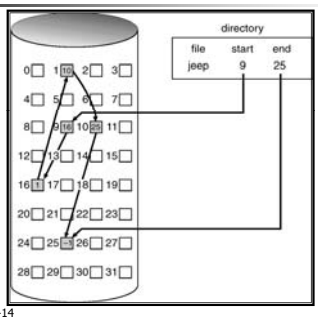
- Each file is a **linked list** of disk blocks:
  - Blocks may be scattered(rải rác) anywhere on the disk.



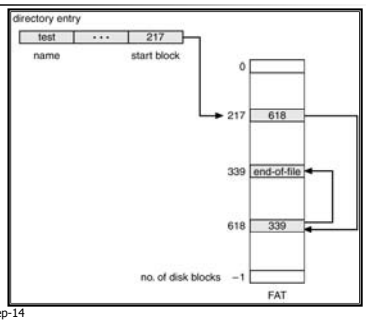
2.3.2. Linked Allocation (2)

- Simple – need only starting address
  - Địa chỉ bắt đầu là con trỏ trỏ đến đầu danh sách liên kết
- Free-space management system – no waste of space
- No random access
- Mapping
  - File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.
  - Truy cập bảng FAT => danh sách các khối của file

2.3.2. Linked Allocation(3)

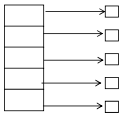


2.3.2. File-Allocation Table(4)

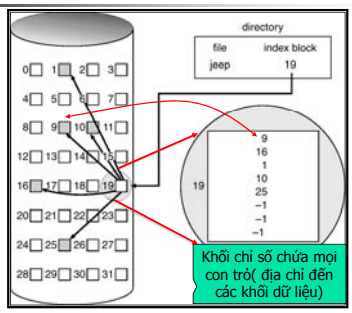


2.3.3. Indexed Allocation(1)

- Brings all pointers together(gắn với) into the *index block.*
  - Mọi con trỏ gắn trong một khối chỉ số xác định
- Logical view.



2.3.3. Example of Indexed Allocation(2)



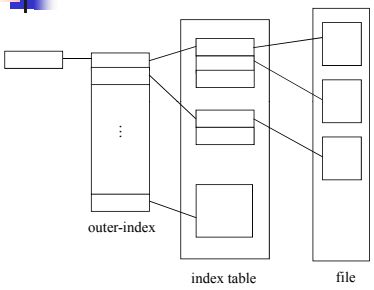
2.3.3. Indexed Allocation (3)

- 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.(Để ánh xạ 1 file có kích thước tối đa là 256K từ nhớ với kích thước 1 khối là 512 từ nhớ*<chứa tối đa 512 địa chỉ đến các block dữ liệu>* chỉ cần 1 khối cho bảng chỉ số)
  - Vì 512 x 512 = 2^9 x 2^9 = 2^8 x 2^{10} = 256K

2.3.3. Indexed Allocation – Mapping (4)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words). (Ánh xạ đến file có kích thước tùy ý)
- Linked scheme – **Link blocks of index table** (no limit on size).
  - Kích thước file tùy thuộc vào kích thước bảng chỉ số của nó

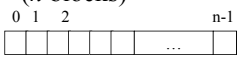
2.3.3. Indexed Allocation – Mapping (5)



- Two-level index (maximum file size is 512^3 = 128M từ nhớ)

2.4. Free-Space Management(1)

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



$$bit[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied(đang sử dụng)} \end{cases}$$

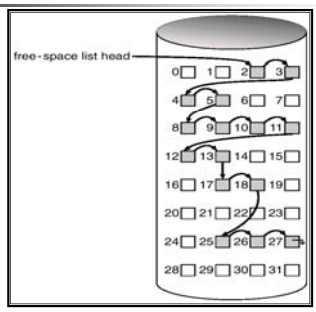
2.4. Free-Space Management (2)

- 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(dễ tổ chức file liên tục)
- Linked list (free list)
  - Cannot get contiguous space easily(khó lấy vùng liên t)
  - No waste of space(không lãng phí)
- Grouping(nhóm các vùng nhớ rỗi)
- Counting(tính toán các vùng nhớ rỗi)

2.4. Free-Space Management (3)

- Need to protect:
  - Pointer to free list
  - Bit map
    - Must be kept on disk(vì cần lưu trữ trạng thái bộ nhớ phụ lâu dài)
    - 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

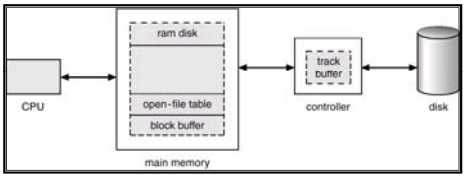
2.4. Linked Free Space List on Disk(4)



2.5. Efficiency and Performance(1)

- Efficiency(hiệu quả) dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance(hiệu năng-tốc độ):
  - disk cache – separate section of main memory for **frequently used blocks**
  - free-behind and read-ahead – techniques to optimize sequential access(tối ưu truy cập tuần tự)
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk.

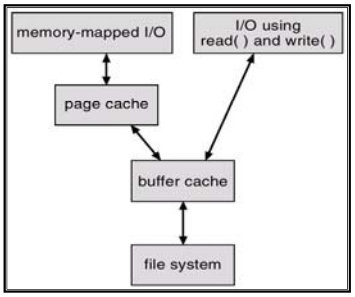
2.5.1. Various Disk-Caching Locations



2.5.2. 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.

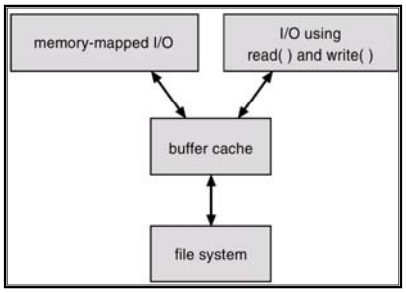
2.5.3. I/O Without a Unified Buffer Cache



2.5.4. Unified Buffer Cache

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

2.5.5. I/O Using a Unified Buffer Cache



2.6. 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).
- Recover lost file or disk by *restoring* data from backup.

2.7. 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.

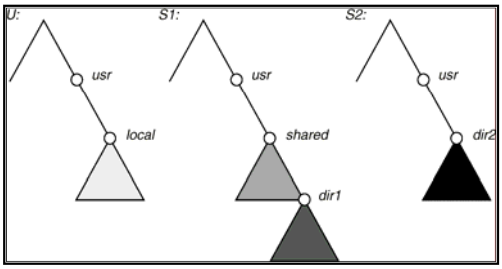
2.8. NFS (1)

- 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, replacing the subtree descending from the local directory.
  - 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.

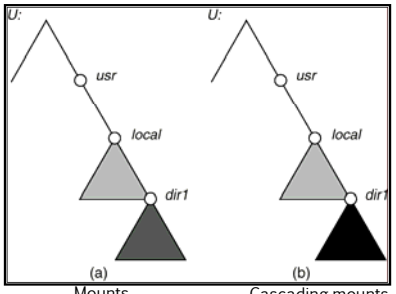
2.8. NFS (2)

- 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.

2.8.1. Three Independent File Systems



2.8.2. Mounting in NFS(1)



2.8.2. NFS Mount Protocol(2)

- 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.

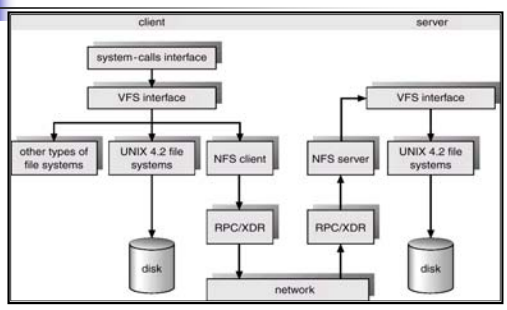
2.8.2. NFS Protocol(3)

- 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 (lose advantages of caching).
- The NFS protocol does not provide concurrency-control mechanisms.

2.8.3. 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.

2.8.4. Schematic View of NFS Architecture



2.8.5. 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.

2.8.6. 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.

Q & A

- List câu hỏi