高级操作系统

Advanced Operating System

Distributed Systems

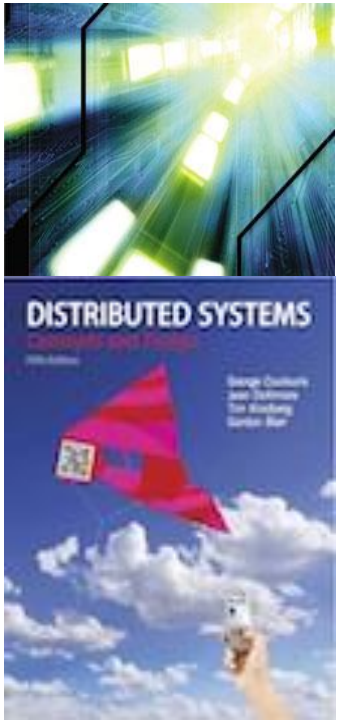
Concepts and design

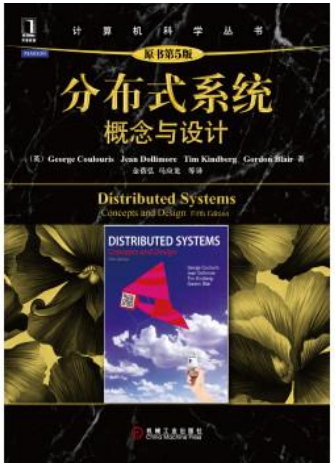
朱 青 **Qing Zhu,**

Department of Computer Science, Renmin University of China

zqruc2012@aliyun.com

Chapter 7 Distributed File Systems
分布式文件系统





朱 青

信息学院计算机系，

中国人民大学，

zqruc2012@aliyun.com

Textbook

- Coulouris, Dollimore, Kindberg and Blair
- Distributed Systems – Concepts and Design,
  George Couloris, Jean Dollimore, and Tim Kindberg
  Edition 4,5 © Addison-Wesley 2012
- The lecture is based on this textbook.

Outline

- 7.1 简介
- 7.2 文件系统操作
- 7.3 文件服务架构
- 7.4 UNIX文件系统操作
- 7.5 分布式文件访问模型
- 7.6 NFS网络文件系统
- 7.7 虚拟文件系统
- 7.8 SUN网络文件系统

简 介

- 文件系统
  - 持久存储（**persistent storage**）
- 分布式文件系统
  - 持久存储
  - 信息共享
  - 类似（有时更好）的性能和可靠性

Storage Systems and Properties (1)

可共享 持久性 缓存/副本 维护一致性

	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	×	×	×	1	RAM
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	✓	Ivy (DSM)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore

Types of consistency:一致性类型
1: strict one-copy. 2: considerably weaker guarantees.
1：严格的单副本 2：相对弱的保障

Storage Systems and Properties (2)

- Sharing 共享
  - Information: Files are shared by several clients.
  - 信息：文件被数个客户共享
  - Resources: Persistent storage is shared by several users
  - 资源：持续的存储被数个客户共享
- Persistence 持续性
  - Information (objects) can be used even after the generating program has terminated.
  - 即便程序终止，信息仍可使用
- Distributed caches/ replicas 分布式缓存/复制
  - There are possibly several copies of a file at different locations from which those copies can be retrieved.
  - 可能有多个文件副本存放在不同的地方，并被检索到。

Storage Systems and Properties (3)

- Consistency 一致性
  - Consistency is strict if the user cannot observe any discrepancies between the original file and a cached copy.
  - 如果用户无法观察到原始文件和缓存中副本的差异，那么就保证了严格的一致性。
  - Some systems adopt various mechanisms to achieve some form of consistency without guaranteeing strict consistency.
  - 一些系统采用了多样的机制来保证一定程度上的一致性，但并不保证严格的一致性

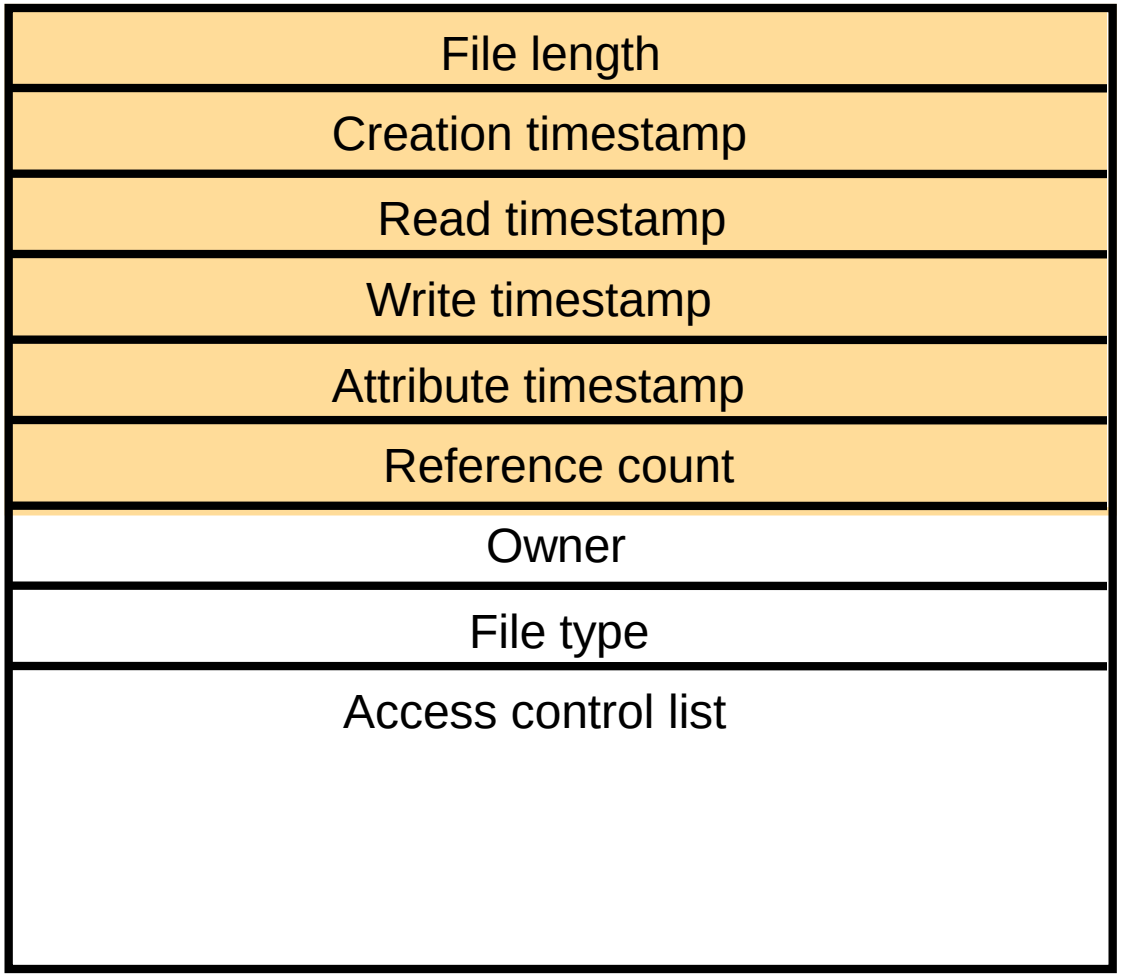
文件系统的特点

- 功能
  - 组织, 存储, 检索, 命名, 共享和保护
- 文件相关的重要概念
  - 文件：包含数据和属性
  - 目录：提供从文件名到内部文件标识映射的特殊文件
  - 元数据：额外的管理信息，包括属性、目录等
- 文件系统结构
- 文件系统操作

File System Modules文件系统模块

Directory module: 目录模块：	relates file names to file IDs 将文件名与文件号关联
File module: 文件模块：	relates file Ids to particular files 将文件号与特定文件关联
Access control module: 访问控制模块：	checks permission for operation requested 检查请求操作的许可
File access module: 文件访问模块：	reads or writes file data or attributes 读/写文件数据或属性
Block module: 块模块：	accesses and allocates disk blocks 访问和分配磁盘块
Device module: 设备模块：	disk I/O and buffering 磁盘I/O及缓存

File Attribute Record Structure文件属性记录结构



文件长度

创建时间戳

读时间戳

写时间戳

属性时间戳

引用计数

拥有者

文件类型

访问控制列表

File Content 文件内容（1）

A file contains data and attributes.文件包含数据和属性

- **Data:数据**:
  Sequence of data items (UNIX: Bytes)数据项目的序列
  - accessible by operations to **read** or **write** any portion of the sequence
  - 通过操作访问，可**读**或**写**序列的任何部分

File Content 文件内容（2）

- **Attributes**属性:
  A single record containing information of the file.包含文件信息的简单记录
  - length of the file, various timestamps (creation read, write, attribute), reference count
  - 文件长度，各种时间戳（创建、读、写、属性），引用计数
    - Those attributes are managed by the file system.
    - 这些属性由文件系统管理
  - owner, file type, access control list
  - 拥有者，文件类型，访问控制列表
    - Those attributes can be updated by the owner.
    - 这些属性可以被拥有者更新

Observed Properties of File Systems
观察到的文件系统属性（1）

- Most files are small (smaller 10 k Bytes).大部分文件很小。（不到10KB）
- Reading is much more common than writing.读操作比写更常见。
- Reads and writes are sequential; random access is rare.
- 读和写是顺序的；随机操作很少见。
  - This is true for most data files.这一条对大多数数据文件是适用的。

Observed Properties of File Systems
观察到的文件系统属性（2）

- Most files have a short lifetime.大多数文件生命周期很短。
  - There are many temporary files.有很多是临时文件。
- File sharing is unusual.文件共享不多。
  - In most cases, shared files are read-only.
  - 大多数情况下，共享的文件是只读的。
- The average process uses only a few files.
- 通常的进程只用很少的一些文件。
- Distinct file classes with different properties exist.
- 不同的文件类型有不同的性质。

File System Operations
文件系统操作（1）

- New files can be created or existing files can be opened.
- 新文件可以被创建；已存在的文件可以被打开。
  - A file descriptor referencing the open file is returned.
  - 返回与打开的文件相关的文件描述符。
- A file can be closed.
- 文件可以被关闭。
- Bytes can be transferred between a file and a buffer.
- 字节可以在文件和缓冲区之间互相传送。
  - This operation advances the read-write pointer for the file.
  - 这个操作推进文件的读写指针。
- The read-write pointer can be moved.

File System Operations
文件系统操作（2）

- 读写指针可以移动。
- A file can be removed from the directory structure.
- 文件可以从目录结构中移除。
  - If a file has no other names (cannot be referenced anymore) the file is deleted.
  - 如果文件没有任何名字（无法再被引用），则删除。
- A new name can be added for a file.
- 可以为文件添加新名称。
- The file attributes can be written to a buffer.
- 文件属性可以被写入缓冲区。

分布式文件系统的需求（1）

- 透明性
  - 访问透明性
  - 位置透明性
  - 移动透明性
  - 性能透明性：当服务负载在一定范围内变化时，客户程序可以保持满意的性能
  - 扩展透明性：文件服务可以扩充，以满足负载和网络规模的增长
- 并发文件更新
  - 并发控制：客户改变文件的操作不影响其他用户访问或改变同一文件的操作
- 文件复制：多个副本
  - 更好的性能与容错
- 硬件和操作系统异构性：文件服务的接口必须有明确的定义。在不同操作系统和计算机上实现客户和服务器软件

分布式文件系统的需求（2）

- 容错
  - 为了处理暂时的通信错误，容错设计可以基于最多一次性语义
  - 对于幂等操作：支持最少一次性语义
  - 无状态的服务器: 崩溃重启时不需恢复
- 一致性
  - Unix提供单一副本更新语义
  - 当文件在不同地点被复制和缓存时，可能会偏离单一副本更新语义
- 安全性
  - 身份验证，访问控制，安全通道
- 效率
  - 应提供比传统文件系统相同或更强的性能和可靠性

分布式文件系统的关键目标

- 如何保证透明性？
- 如何保证性能？
- 如何保证容错？
- 如何保证并发操作？

Transparency Requirements
透明性要求

- **Access transparency** 访问的透明性

  A single set of operations is provided for access to local and remote files.提供了一个单一的操作集合来访问本地和远程的文件。

- **Location transparency 位置的透明性**

  Client programs should see a uniform file name space.

  用户程序看到的是一个统一的文件名空间。

  - Files or group of files can be relocated without changing their pathnames.
  - 文件或文件组可以重新部署，无需改变他们的路径名称。

- **Mobility transparency**移动的透明性

  Neither client programs nor system administrator tables in client nodes must be changed when a file is moved.

  当文件被移动后，用户程序和用户节点的系统管理员表均不需更改。

- **Performance transparency 性能的透明性**
  The client performance should be satisfactory even if the system load changes within a reasonable range.
  即使系统负载在合理范围内变动了，仍应为用户提供满意的性能。
- **Scaling transparency**伸缩的透明性
  The service can be incrementally expanded to address a wide range of loads and network sizes.
  服务可以被逐步扩展到可以适用于一个更大范围的负载及网络规模。

Distributed File Systems Requirements分布式文件系统需求

- **Concurrent file updates 并发文件更新**

  Changes to a file by one client should not interfere with operations on this file by other clients.

  一个用户对文件的更改不应影响其他用户对该文件的操作。

  - Possible use of file locking mechanisms
  - 可以使用文件锁机制
- **File replication including consistency文件复制一致性**
- **Hardware and OS heterogeneity硬件和操作系统异构**

  The service interfaces should not be restricted to a single or a few operating or hardware systems

  服务界面不应受限于少数一些操作系统或硬件系统

- **Fault tolerance 容错性**
- **Security安全性**

  This is frequently achieved by access control lists.

  通常是通过访问控制表来实现的。
- **Efficiency效率**

  The performance of a distributed file system should at least match the performance of a conventional file system.

  分布式文件系统的性能应至少和常规的文件系统相当。

文件服务系统结构

文件服务的三个模块：

- 平面文件服务 （**Flat file service)**
  - 对文件内容进行操作
  - 文件唯一标识符 (Unique file identifier (UFID))
- 目录服务（**Directory service**）
  - 生成目录
  - 在目录中加入新文件名
  - 提供从文件名到**UFID**的转换，客户通过向目录服务提供文件名获得文件的**UFID**
  - 是平面文件服务的客户
- 客户端模块
  - 提供单一应用程序接口，支持应用程序透明地存取远程文件服务

File Service Architecture (1)
文件服务架构（1）

A file service consists of 3 components.

文件服务由三部分组成：

- **Flat file service** 平面文件服务
- It implements operations on the content of individual files. Unique file identifiers (UFID) are used to refer to files in all flat file service operations.
- 在独立的文件的内容上执行操作。 在执行所有平面文件服务操作时，使用UFID来指明文件。

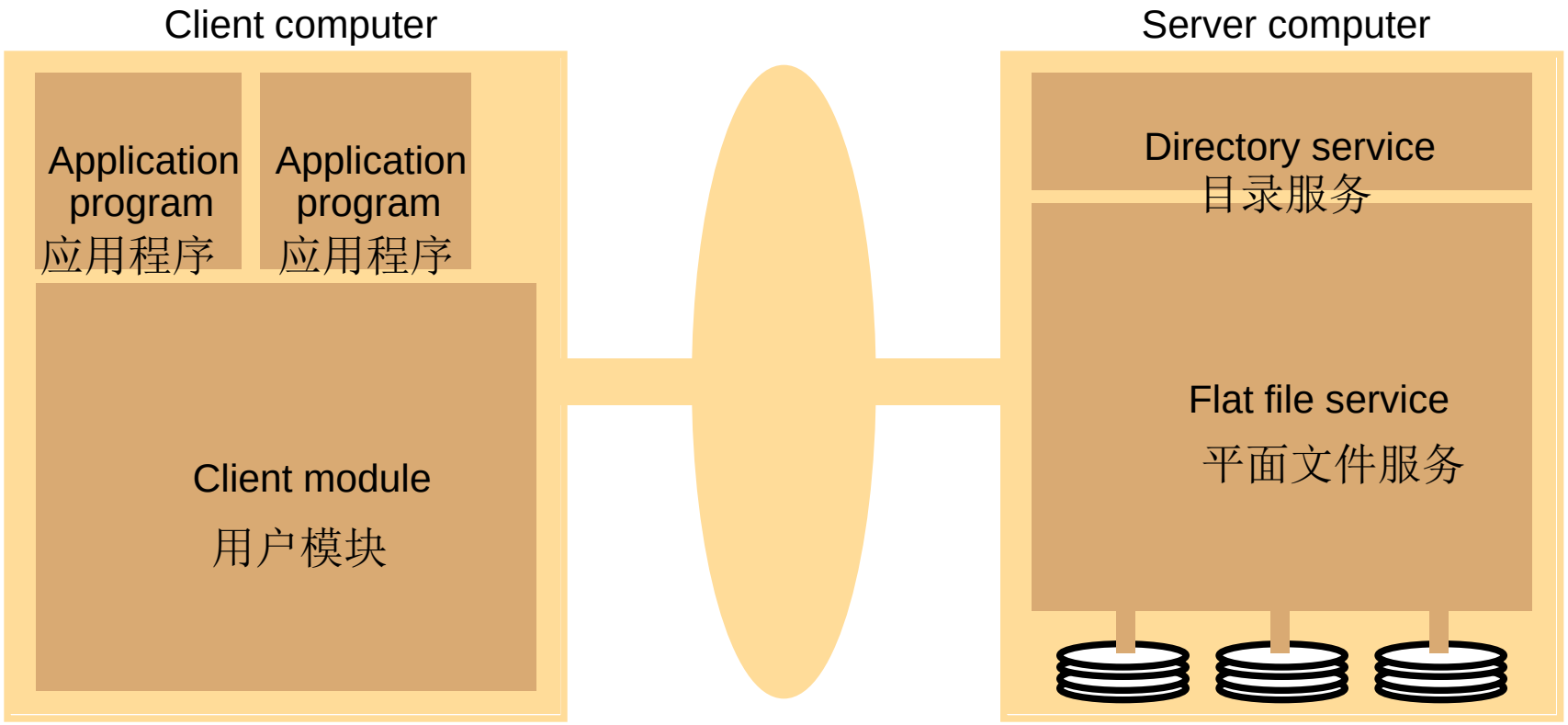
File Service Architecture (1)
文件服务架构（1）

- **Directory service**目录服务
  - It maps text names to unique file identifiers UFID. It is a client of the flat file service.
  - 将文本名称映射为UFID。是平面文件服务的用户。
  - UFID is interface between flat file and directory service
  - UFID是平面文件和目录服务的接口。

Client module用户模块

- It integrates the operation of the other services under a single application programming interface.
- 在单一的应用程序界面下整合了其他服务的操作。
- The client module runs in every computer while the other components only run in the file server.
- 用户模块在每个计算机上运行，而其他模块只在文件服务器上运行。

File Service Architecture (2)



Flat File Service (RPC) Operations

- *Read(FileId,i,n)⌫Data*

  If 1≤*i*≤*Length(File)*: reads sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.

  如果1≤*i*≤*Length(File)*，从第i个项目开始，读入最多n个项目，并用*Data*返回。

- *Write(FileId,i,Data)*

  If 1≤*i*≤*Length(File)*: writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.

  如果1≤*i*≤*Length(File)*，从第i个项目开始，将序列*Data*写入文件，必要时可扩展文件。

- *Create()⌫FileId*

  Creates a new file of length 0 and delivers a UFID for it.

  创建一个长度为0的新文件，并为他分配一个UFID。

- *Delete(FileId)* — Removes the file from the file store.
  从文件存储器中移除文件。
- *GetAttributes(FileId)⌧Attr* — Returns the file attributes for the file.
  返回文件属性。
- *SetAttributes(FileId, Attr)* — Sets the file attributes (only those attributes that can be changed by the user).
  设置文件属性（只有那些可被用户修改的属性）

All of those operations (except *Create*) throw exceptions if the *FileId* argument contains an invalid UFID or if the user does not have sufficient access rights.

当域变量包含一个非法UFID，或当用户没有充分的访问权限时，以上所有的操作（除了Create）都会抛出异常。

平面文件服务接口

- 与Unix比较
  - 无open和close操作
  - Read write操作执行于指定的开始点
- 原因：出于容错的考虑
  - 可重复的操作
    - 除了create，所有的操作都是幂等的，允许至少一次的RPC语义
  - 无状态服务器
    - 文件操作无指针
    - 崩溃重启时不需恢复

Servers With and Without States 状态

Stateless servers无状态的服务器

- Fault tolerance容错性
- No OPEN/CLOSE calls needed
  不需要OPEN/CLOSE调用
- No server space wasted on tables不会在表上浪费服务器空间
- No limits on number of open files不限制打开的文件数目
- No problems if a client crashes当一个客户荡机的时候不会有问题

Stateful servers有状态的服务器

- Shorter request messages更短的请求消息
- Better performance更好的性能
- Readahead possible可以预先读入
- Idempotency easier更加容易的幂等性
- File locking possible可以对文件加锁

Comparison比较

- RPC operations RPC操作
  - All operations with the exception of *Create* are idempotent.
    除*Create*外，所有的操作都是幂等的.
    - Create produces a different file with each call.
    - 每次调用Create都会产生不同的文件。
    - At-least-once RPC semantics is supported.
    - 至少有一次，RPC语义是被支持的。
  - The interface can be implemented by stateless servers. 界面可以由无状态的服务器实现。

- It can be restarted after a failure without any need for clients to restore any state.
- 失败后可以重启，不需要客户恢复任何状态。

A stateless implementation is not possible.无状态的实现是不可能的。

- UNIX operationsUNIX操作

Some operations are not idempotent as a read-write pointer is used.由于使用了读写指针，一些操作不是幂等的。

- A repeated write operation writes data on a different part of the file.重复的写操作将数据写入文件的不同部分。

UNIX File System Operations

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

UNIX文件系统操作

filedes = open(name, mode) *filedes = creat(name, mode)*	用指定的名称打开已存在的文件。 用指定的名称创建新文档。 这两个操作都会返回与打开的文件相关联的文件描述符。 模式有读、写或读写。.
status = close(filedes)	关闭已打开的文件 *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	将 *n*个字节从被 *filedes* 引用的文件传送到*buffer*. 将 n个字节从*buffer*传送到被 *filedes* 引用的文件. 这两个操作都会返回实际传送的字节数目并且会推进读-写指针。
pos = lseek(filedes, offset, whence)	将读-写指针移动至offset (由*whence*的值决定是相对的，还是绝对的).
status = unlink(name)	将文件 *name* 从目录结构中移除。如果文件没有其他的名字，则等同于被删除.
status = link(name1, name2)	给文件 (*name1*)添加新名字(name2) .
status = stat(name, buffer)	将文件*name*的文件属性读入buffer.

Directory Service (RPC) Operations
目录服务操作

- *Lookup(Dir, Name)* ⌫*FileId*
  Locates the text name in the directory and returns the relevant UFID.
  在目录中定位文本名称，返回相关的UFID
- *Add Name(Dir, Name, File)*
  If *Name* is not in the directory, adds (*Name,FileId*) to the directory and updates the file’s attribute record.
  如果Name不在目录中，将（Name，FileId）添加到目录中，并更新文件的属性记录。

- *UnName(Dir, Name)*
  If *Name* is in the directory then the entry containing *Name* is removed from the directory.
  如果Name在目录中，那么包含Name的条目将从目录中移除。
- *GetNames(Dir, Pattern)* ⌫*NameSequence*
  Returns all the text names in the directory that match the regular expression *Pattern*.
  返回目录中所有能够匹配正则表达式Pattern的文本名称。

If an operation is not successful then an exception is thrown.
如果一个操作不成功，则抛出异常。

Directory Service Interface目录服务界面

The directory service has the primary purpose to translate text names to UFIDs.

目录服务主要的目的是将文本名称转换成UFID

- A directory contains mapping between text names for files and UFIDs.目录包含文件的文本名称和UFID之间的映射、
- Each directory is a conventional file.每个目录都是一个常规的文件。
- The directory service is a client of the file service.目录服务是文件服务的用户。

- Operations that alter directories also change the reference count field in the file’s attribute record.
- 更改目录的操作同时会更改文件属性记录中的引用计数域。
  - *AddName* increments the reference count field.
  - AddName增加引用计数。
  - *UnName* decrements the reference count field.
  - UnName减少引用计数。
  - The file is removed if the reference count field reaches the value 0.当引用计数达到0时，文件被移除。
    - Then the file cannot be reached anymore.文件就无法再被访问到了。

层次性文件系统

- 目录树
  - 每个目录是一个特殊文件
    - 存储包含的文件和其他目录的名字
  - 路径名
    - 引用文件和目录
    - 多部分名，如“/etc/rc”
- 在目录树中查找
  - 借助多个Lookup操作翻译路径名
  - 客户端的目录缓存

Hierarchic File System层次文件系统

A hierarchic file system consists of a number of directories arranged in a tree structure.

层次文件系统包含许多以树形结构排布的目录。

- Each directory holds the names of the files and directories that are accessible from it.
- 每个目录保存着文件的名称和可以从该目录访问到的目录名。
- Any file or directory can be referenced with a **pathname**.
- 每个文件或目录可以通过路径名称被提及。
  - A sequence of names that represent a path through the tree
  - 名称的序列，代表着穿过树的一条路径

- Every part in a pathname, except possibly the last, refers to a directory.
- 路径名称除了最后一部分外的每一部分都涉及到一个目录
  - The attributes of a file should distinguish between ordinary files and directories.
  - 文件的属性应该与普通文件及目录区分开来。
- The root of the tree has a distinguished name.
- 树的根应该有一个能区分开来的名字。

- The UNIX file system is not a strict严格的 hierarchy 层次.
- UNIX文件系统并没有严格的层次。
  - Files can have several names by use of the *link* operation.
  - 通过使用link操作，文件可以有多个名字。

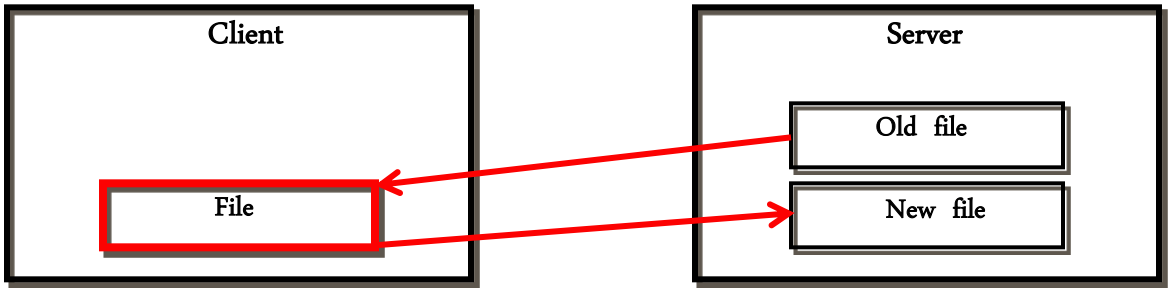
分布式文件系统访问控制

- Unix文件系统
  - 系统会将用户的访问权限与 open 调用要求的模式（读或写）相比较
- 无状态的分布式文件系统
  - 访问权限在服务器进行
  - 访问控制的方法：
    - 基于权能的验证（权能作为以后一系列的访问许可被返回给客户）
    - 每次请求都发送用户ID，每次文件操作，服务器都进行访问检查
    - AFS（Andrew文件系统） 和 NFS中的Kerberos

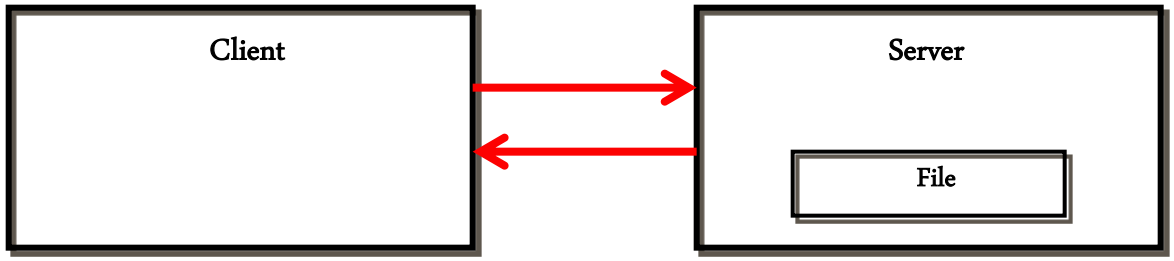
Access Models (1)访问模型

1. File is moved to the client文件被移动到客户机上
2. File is accessed on client文件被客户访问。
3. File is moved to the server文件被移动到服务器上

Upload/Download Model
上传/下载模型



Remote Access Model
远程访问模型



Access Models (2)访问模型

Upload/Download Model上传/下载模型

- File manipulation is only done at the client.
- 文件处理只发生在客户端。
- Whole files are moved to and from the client.文件整体移动到客户端或从客户端移动。
  - Transfer is efficient but potentially wasteful.
  - 传输是有效率的，但存在潜在的浪费。
  - Enough storage is required at the client.
  - 客户端需要有足够的存储空间。

- The file system only performs read and write operations. 文件系统只进行读写操作。
  - The implementation is simple.实现简单。

Remote Access Model远程访问模型

- File manipulation is done on the server.
- 文件处理发生在服务器端。
- No transfer of the whole file is necessary.
- 不需要传输整个的文件。
- Potentially a large number of messages is required.
- 传输大量消息是潜在的需要。

Access Control访问控制

- In the UNIX file system, the user’s access rights are checked against the access *mode* in the *open* call. The file is only opened if the user has the necessary access rights.在UNIX文件系统中，当处在open调用的访问模式时，需要检查用户的访问权限。只有当用户有相应的访问权限时，文件才会被打开。
  - The access rights are maintained until the file is closed.
  - 在文件关闭前，访问权限一直被维持。
  - No further checks are required.
  - 不需要更多的检查。

- In a distributed file system, access rights must be executed at the server to prevent unauthorized access via the RPC interface.
- 在分布式文件系统中，访问权限必须在服务器端执行，以防止通过RPC界面进行的非授权的访问。

- Stateful server: The result of an access rights check is retained at the server.
- 有状态的服务器：访问权限检查的结果在服务器端保存。
- Stateless server:无状态的服务器：
  - An access rights check is made whenever a file name is converted to a UFID. The results produce a capability性能 that is returned to the client.
  - 每当文件名被转换成UFID时，都要进行访问权限检查。检查的结果生成可能性，并返回客户端。
  - A user identity is submitted with every client request.
  - 用户标识和每个用户请求一并被提交。

Network File System NFS网络文件系统

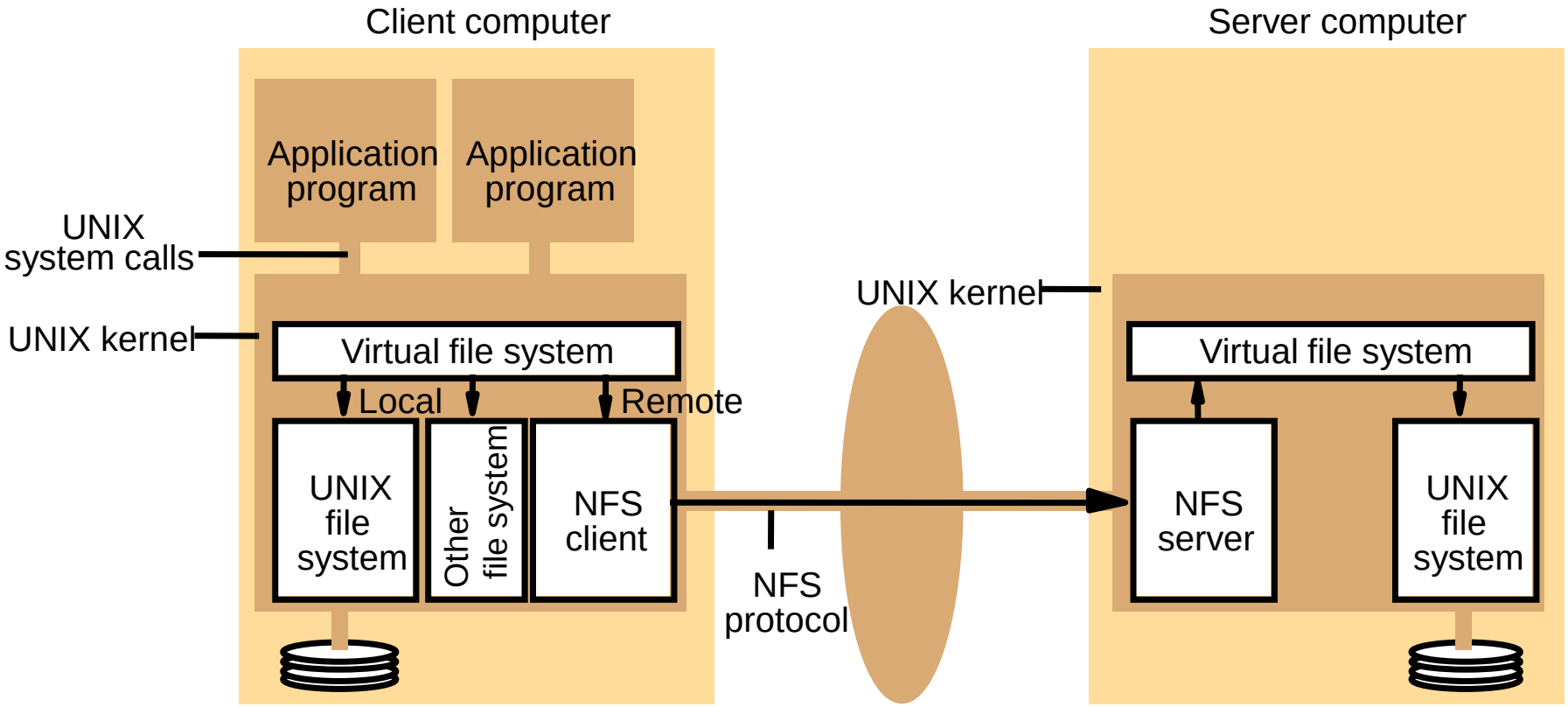
NFS has been introduced by Sun Microsystems as one of the first commercial distributed file systems.

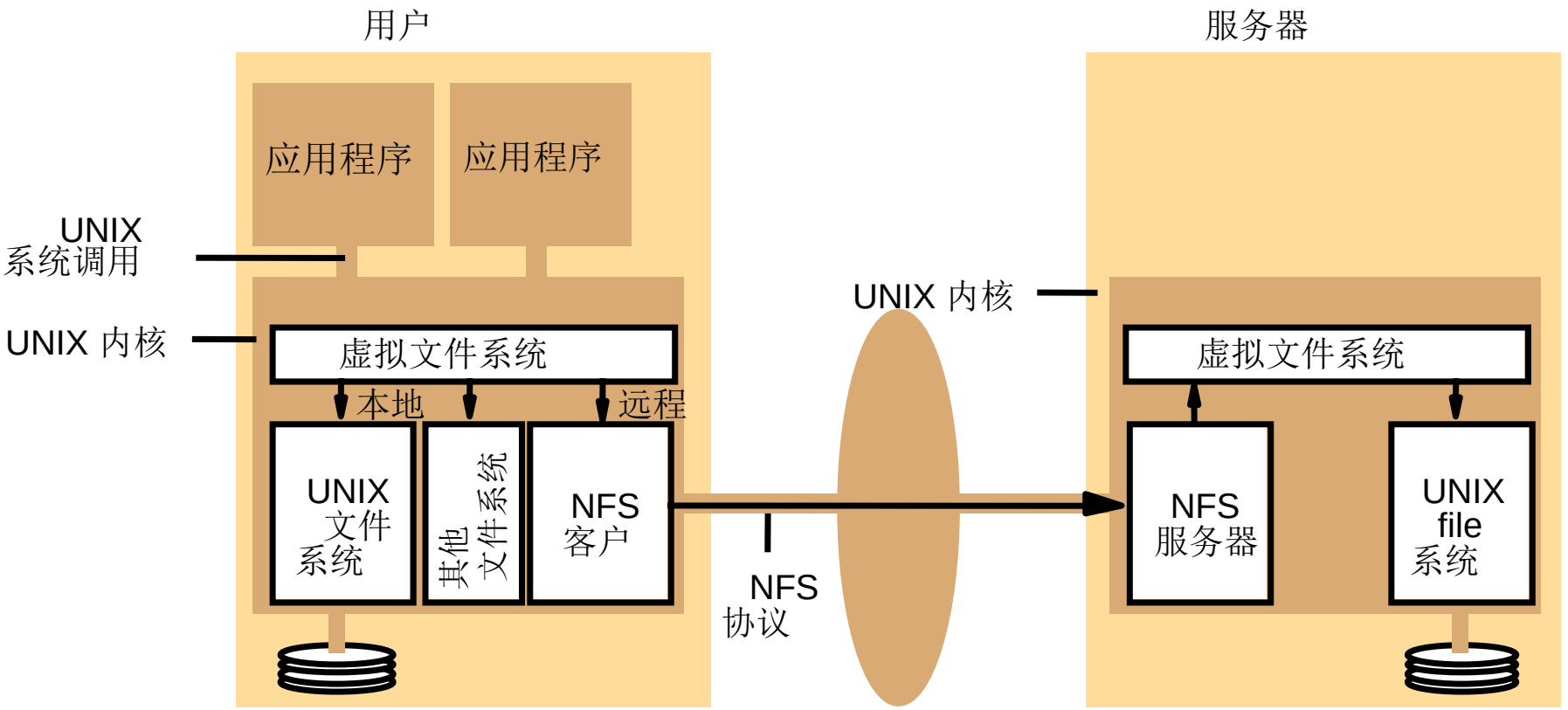
NFS是Sun公司引入的第一个商业化的分布式文件系统。

- It is frequently considered to be the *de facto*事实上 standard.它通常被认为是事实上的标准。
- NFS supports heterogeneous distributed systems:NFS支持不同种类的分布式系统：
  - different operating systems and different machine hardware不同的操作系统和不同的机器硬件

- The NFS client and server modules are typically installed in the kernel of the OS on each machine in the network.
  - Each machine can be both an NFS client and an NFS server at the same time.
- Communication between server and client is done via RPC.
- Access of programs to files is executed via system calls and does not require recompilation再编辑 of reloading再安装.
  - One client module serves all user-level processes and has a shared cache.
  - Authentication鉴定 is done in the kernel.

NFS Architecture　NFS架构




用户
服务器
应用程序
应用程序
UNIX
系统调用
UNIX 内核
虚拟文件系统
本地
远程
UNIX
文件
系统
其他
文件系统
NFS
客户
NFS
协议
UNIX 内核
虚拟文件系统
NFS
服务器
UNIX
file
系统

Virtual File System 虚拟文件系统

NFS implements a virtual file system.

NFS实现了一个虚拟文件系统。

- The virtual file system distinguishes between local and remote files.
- 虚拟文件系统区分了本地文件和远程文件。
- It translates between NFS **file handles** and remote identifiers.
- 它负责NFS文件句柄和远程标识之间的互译。
- It keeps track of available (sub-) file systems.
- 它可以明示出可用的（子）文件系统
- There is one v-node per open file in the virtual file system layer.在虚拟文件系统层，每个打开的文件都有一个v-node。

- The file handle consists of
- 文件句柄包含
  - a file system identifier,
  - 文件系统的标识符
  - an i-node number of file (in UNIX),
  - 文件中含有i-node的数目
  - an i-node generation number (in UNIX).
  - i-node的一代的数目
    - The generation number handles the re-use of i-node numbers.
    - 一代的数目控制了可重用的i-node数。

File Export 文件输出

A client can only access files remotely that have been exported before by a server.

客户只能访问那些已被服务器输出的远程文件。

- An NFS servers usually exports directories automatically when it is booted.
- NFS服务器通常会在启动的时候自动输出目录。

- Export rules for servers:
  - Remote file systems cannot be exported.
  - Each local file system and each subset of it can be exported.
  - A directory and one of its (direct or indirect) subdirectories cannot be exported at the same time unless they reside on a different physical device.
  - No file can be exported twice at the same time.

Directory and File Access　目录和文件访问

An NFS server is **stateless**.NFS服务器是无状态的。

- NFS supports normal file access operations **read, write, getattr, setattr** but not the UNIX system calls OPEN and CLOSE.
- NFS支持标准的文件访问操作：read，write，getattr，setattr。但是不支持UNIX调用OPEN和CLOSE
  - NFS supports other operations on directories (similar to UNIX) like **mkdir, rmdir**,….
  - NFS还支持其他的目录操作，像mkdir，rmdir……

- Each message from a client to a server is self contained.
- 每个从客户机到服务器的消息都是自包含的。
- Locks are not supported. 不支持锁。
- A server crash and later reboot will not result in lost information about open files.
- 服务器荡机并重启不会导致打开的文件丢失信息。
- NFS uses the UNIX protection mechanism
- NFS使用UNIX的保护机制。
  - rwx for owner, owner’s group and others.
  - rwx给用户、用户所在的组和其他人。

NFS Server Operations (simplified) (1)

lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*. 返回文件句柄和名为name的文件的属性，该文件在dirfh目录下。
create(dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes. 在目录dirfh下用属性attr建立一个叫name的新文件，返回新文件的句柄和属性。
remove(dirfh, name) status	Removes file name from directory *dirfh*. 从目录dirfh下移除文件name。
getattr(fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.) 返回文件fh的文件属性。（与UNIX stat的系统调用相似。）

Continues on next slide ...

setattr(fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file. 设定属性（模式，用户id，组id，大小，访问时间，修改时间）。将大小设为0将截断文件。
read(fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file. 从一个文件的offset处起返回count字节的数据。同时返回文件的最新属性。
write(fh, offset, count, data) -> attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place. 将文件从offset处开始写入count字节的数据。当写入完毕时返回文件属性。
rename(dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh* 将目录dirfh中的文件name重命名为目录todirfh中的文件toname。
link(newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*. 将目录newdirfh中的newname链接到目录dirfh中的name文件。

NFS Server Operations (simplified) (2)

symlink(newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it. 创建目录newdirfh下的名为newname的入口，用string表示它的象征链接的类型。服务器并不翻译string，而是用象征链接的文件保存string。
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*. 返回被fh标识的象征链接文件所关联的string。
mkdir(dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes. 创建属性为attr的新目录name，并返回新的文件句柄和属性。
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty. 从父目录dirfh下移除名为name的空目录。如果该目录非空，则失败。

readdir(dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory. 返回目录dirfh中至多count字节的目录入口。每一个入口包含一个文件名、一个文件句柄，一个指向下一个目录入口的不透明指针（称为cookie）。cookie用在后继的readdir调用中，使调用从紧接着的入口开始读入信息。如果cookie的值为0，那么从目录的第一个入口开始读入信息。
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*. 返回包含文件fh的文件系统信息（比如块大小、空闲块的数目等等）

Mounting 加载(1)

There is a separate mount service process at the user level of each NFS server.

在每个NFS服务器的用户级别上，有一个独立的加载服务进程。

- A client sends a request to a server and asks to mount the specified directory. This request includes
- 每个客户发送请求到服务器，要求加载特定的目录。请求包括：
  - the remote host system,远程主机系统
  - the pathname of a directory, and目录的路径名称
  - the local name for mounted directory.要安装的目录的本地名称

- The server returns a file handle (via VFS layer) containing服务器返回文件句柄（通过VFS层）包括
  - the file system type,文件系统类型，
  - the disk,磁盘，
  - the i-node number of directory (UNIX), and目录的i-node数
  - security information.安全信息。
- File mounting can be done with the help of shell scripts and does not necessarily require any user interaction.
- 文件加载可以在shell脚本的帮助下实现，可以完全不需要任何的用户交互。

Mounting (2)

A file mounting request fails, if

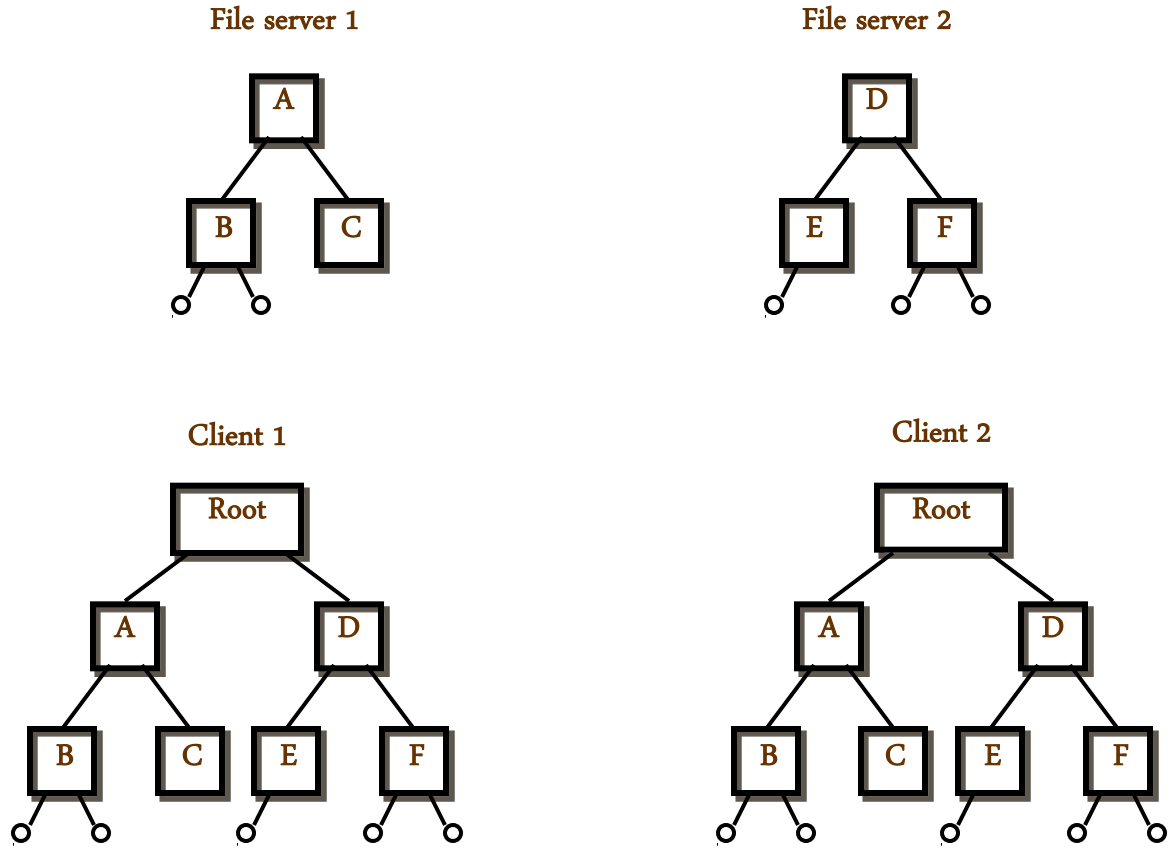
在如下情况下，文件安装请求失败

- the server is not available,
- 服务器不可用
- the request file system has not been exported by the server, or
- 请求的文件系统没有被服务器输出，或
- the client does not have access permission for the requested file system.
- 用户没有对请求的文件系统的进入权限。

Remote file systems can be hard or soft mounted.
远程文件系统可以被硬加载或软加载。

- Soft mounting:软加载：
  - The client returns an error message to the application program after mounting failed a predetermined number of times.
  - 加载失败到预先约定的次数后，客户端返回一条错误信息给应用程序。
- Hard mounting:硬加载：
  - The client repeats the mounting request until it succeeds.
  - 客户端重复安装请求直到安装成功。

Example Views of the Client (1)

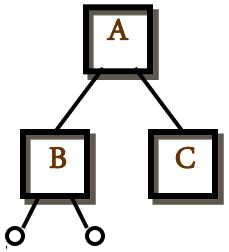


Same view for all clients

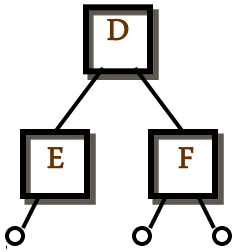
对所有的客户是同样的视图

Example Views of the Client (2)

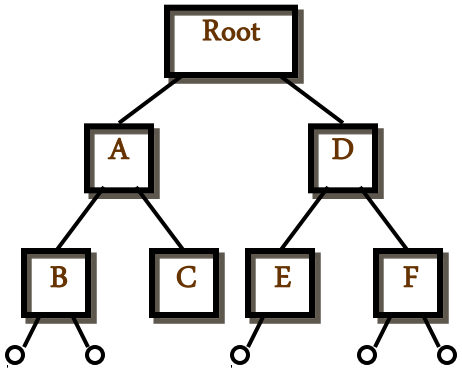
File server 1



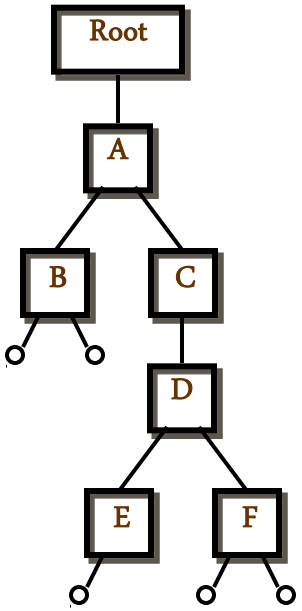
File server 2



Client 1



Client 2



Potentially different view for some clients

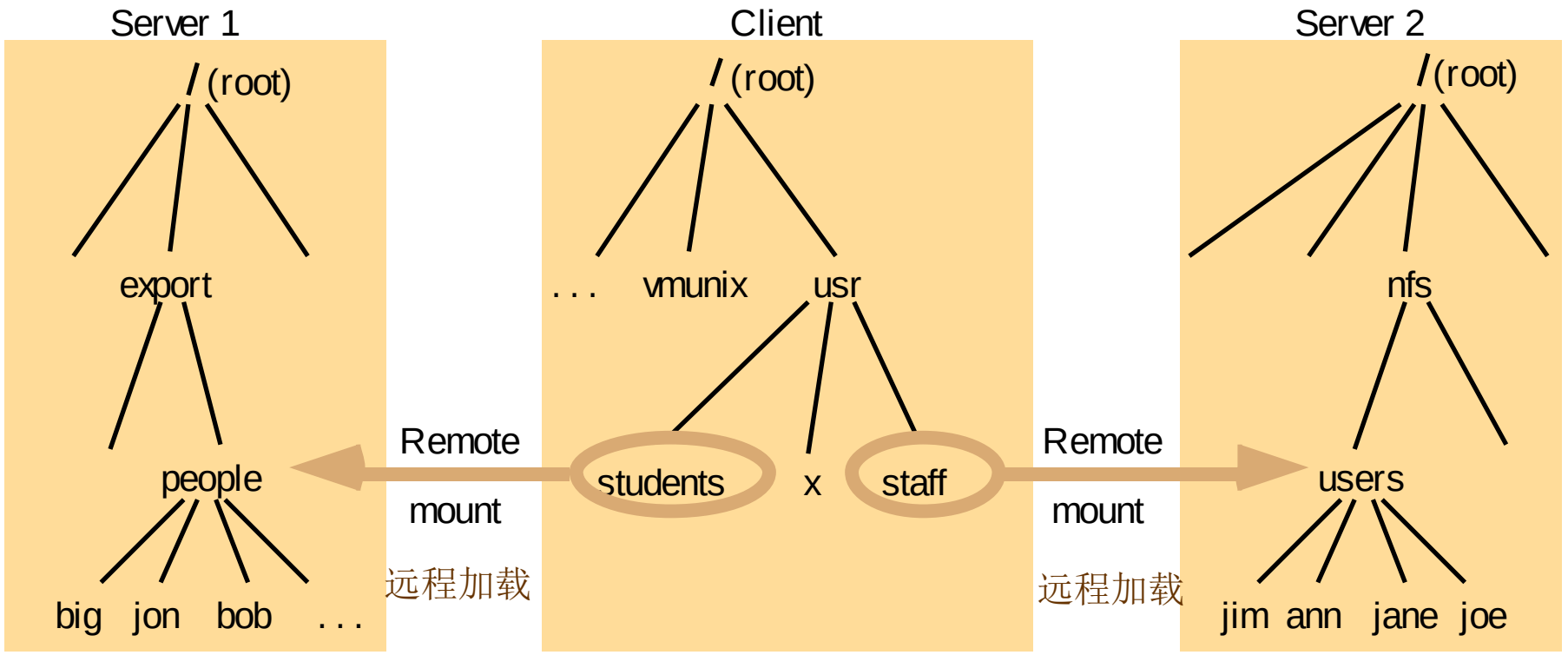
对一些客户有潜在的不同的视图

Views of the Client (3)

1) All clients have the same view of the distributed file system.所有客户对分布式文件系统有相同的视图。
   - Each path is either valid on all machines or on no machine.
   - 每条路径要么对所有的机器都有效，要么都无效。
2) Clients may have different views of the distributed file system.客户对分布式文件系统可能会有不同的视图。
   - This approach is typical for systems with remote mounting.
   - 这种方式对远程加载的系统是典型的。

- Remote mounting 远程加载
  - The remote part in the file system is linked to a selected position in the local file system.
  - 文件系统的远程部分被链接到本地文件系统一个指定的位置。
  - This approach does not provide mobility transparency.
  - 这种方式不提供移动透明性
    - The structure of the file system depends on the machine.
    - 文件系统的结构依赖于机器。

Local and Remote File Systems Accessible on an NFS Client



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1;
the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.
注意：在客户端*/usr/students* 被加载的文件系统，实质上是server1中位于/export/people/的子树。
在客户端/usr/staff被加载的文件系统，实质上是server2中位于/nfs/users的子树。

Automounting

Automounting is an alternative to static mounting as described previously.

自动加载是静态加载的可选办法。

- Each client has a table of mount points.
- 每个客户都有一张加载点的表。
  - reference to one or more NFS servers
  - 参照了一个或多个NFS服务器
- The availability of those directories is not checked when the client is booted.
- 当客户机启动时，这些目录的可用性是不检查的。
  - Booting is possible even if not all servers are available.
  - 即便不是所有的服务器都可用，仍可以启动。

- When the client tries to access a file in such a remote directory, a message is sent to all servers containing this directory.
- 当客户机试图访问远程目录下的一个文件时，将发送消息到所有包含此目录的服务器。
- The directory of the first server who replies is mounted by the client.
- 第一个回应的服务器的目录将被客户加载。
  - Fault tolerance and a crude way of load balancing.
  - 容错性以及天然的负载平衡方式。

- If the symbolic link is not used for some time, the remote directory is unmounted.
- 如果象征链接一段时间没有被使用，那么远程目录将被卸载。
- Automounting is often used for read-only files as NFS does not provide file replication service.
- 自动加载通常用于只读文件，因为NFS不提供文件复制服务。

File Storage 文件存储

A file may be stored at four places.文件可存在四个地方

- Disk of the server (main location)服务器的磁盘（主要位置）
  - Easy to implement 易于实现
  - No consistency problems 没有一致性问题
  - Low performance 性能低
- Main memory of the server 服务器主存
  - Better performance (fewer disk accesses) 性能较好（磁盘访问较少）
  - Block size caching is possible. 可以有块大小的缓存
  - No consistency problems 没有一致性问题
- Disk of the client (if available) 客户机磁盘（如果可用）
  - Useful only for slow networks and little main memory of client
  - 仅当网速慢且客户机内存不多时有用
- Main memory of the client客户机主存
  - Network accesses are avoided.无法通过网络访问。

NFS Server Caching NFS服务器缓存

- NFS uses the main memory of the server to cache file pages, directories, and file attributes.
- NFS使用服务器的主存来缓存文件页面、目录和文件属性
  - reduction of disk accesses减少磁盘访问
- Read ahead 向前读
  Pages that follow those being just read are fetched 紧跟着刚被读到的页面的页面被取出
  - typically 8k packets 通常是8k的信息包

- Delayed write 延迟写

  When a page has been changed, its content is written to disk only if buffer space is required. The buffer is flushed every 30 to guard against system crash.

  当页面被更改时，仅当需要缓冲区空间时，才会将页面内容写入磁盘。缓冲区每30s刷新一次，以防止系统荡机。

  - The client may send a commit operation.
  - 客户机可发送一个提交操作。
  - A reply is sent after data has been written to disk.
  - 当数据被写回磁盘后，发送回复。

NFS Client Caching NFS客户缓存

Client caching is used and may result in inconsistent copies of a shared file.

使用了客户缓存，这可能会导致共享文件不一致的副本。

- A cache block is discarded after a freshness interval (3 - 30 sec) has expired.
- 当刷新间隔（3-30秒）过期后，缓存块被丢弃。
- When opening a cached file, a modification time stamp is requested from the server.
- 当打开一个缓存的文件时，要向服务器端请求一个修改时间戳
- Modified cache blocks are dirty and are sent to the server after the expiration of a timer or when the file is closed.
- 当计时器过期后，或文件已被关闭时，修改过的缓存块是脏的，并被发送到服务器端。

Cache Consistency (1) 缓存一致性

There are four different approaches to address cache consistency, if client caches are used and not all files are immutable.

当客户缓存启用，且不是所有的文件是不变的的时侯，有四种不同的方式保证缓存一致性。

- Write through 始终写
  - Every writing of a file in the cache is immediately sent to the server as well.
  - 对在缓存中的文件的每一次写操作都立即发送到服务器。
    - The most recent file content is on the server.
    - 最进的文件内容在服务器端。

- Another process using a file in the cache may get an old copy.
- 另一个使用在缓存中的文件的进程可能会拿到旧的副本。
  - Server check by cache manager is required.
  - 需要服务器端通过缓存管理器进行检查。
- This approach results in a significant amount of network traffic.
- 这种方式将导致重大的网络通信量。

- Centralized control 中心控制
  - The file server keeps track of all open files. When a client requests to open a file, this request can be granted, queued or denied. The server may send unsolicited messages to request the closing of a file by some client.
  - 文件服务器记录了所有打开的文件。当客户请求打开一个文件时，请求可以被准予、加入队列或否决。服务器可能会发送未请求的消息来要求某些客户关闭文件。
    - Deviation from client/server concept and poor scaling
    - 背离了客户/服务器概念，扩展性不好

Cache Consistency 缓存的一致性 (2)

- Delayed write 延迟写
  - Write updates are collected and only sent to the file server about every 30 seconds.
  - 收集写更新，每30秒才会向文件服务器发送一次
    - Transferring a single big message is more efficient than many small messages.
    - 传送一个单一的大消息比传送许多小消息更有效率。
    - Some temporary files may not be transmitted at all.
    - 一些临时文件可能根本不会被传送。
    - The result of a read is not necessarily deterministic as it depends on the timing.
    - 读的结果依赖于时间，是不确定的

- Write on close 关闭才写
  - A file is only written back to the server after it has been closed.只有当文件关闭的时候才写回服务器端。
    - Session semantics are enforced.
    - 增强session语义
    - Improvement by waiting some time before transmitting the file (combination of delayed write).
    - 在传送文件前等待一些时间，从而得到改进。（与延迟写结合）
    - If two processes write the same file in quick succession, one file gets lost.
    - 如果两个进程快速交替的写同一个文件，一个文件会丢失。

NFS Summary　NFS总结

- Access transparency: use of the virtual file system
- 访问透明性：使用虚拟文件系统
- Location transparency: A uniform name space can be obtained with specific configuration tables
- 位置透明性：从特定的配置表中可得到统一的名空间
  - no different mount points
  - 没有不同的加载点
- Mobility transparency: Mount tables must be updated after migration.
- 移动透明性：移动后需更新加载表。
- Scalability(可测量性): no automatic replication and no support of replication with updates
- 可测量性：没有自动的复制，不支持带更新的复制

- Hardware and software heterogeneity异构
- 硬件和软件异构
- Fault tolerance: NFS is stateless and the operations are idempotent幂等的.
- 容错性：NFS是无状态的，且操作是幂等的。
- Consistency: It comes close to a one copy semantic.
- 一致性：与单副本语义很相近
- Security: NFS does not support encryption but can be extended to include encryption.
- 安全性：NFS不支持加密，但可被扩展为包含加密术

File Replication (1)　文件复制

A distributed file system may offer file replication as an additional feature.

分布式文件系统可能会提供文件复制作为它的附加特性

File replication is typically implemented for several reasons:

文件复制特别的被实现，有以下几个原因：

- **Fault tolerance**:容错性：
  - If a storage device or a server is permanently damaged, data are not lost.
  - 即便一个存储设备或服务器或时常发生损毁，也不会遗失数据。

- **Availability**:可用性：
  - If a server goes down temporarily临时, its data are still available to the client.
  - 如果服务器临时荡机，它的数据对客户机仍可用。
- **Performance**:性能：
  - If a large number of clients tries to use the same server or even the same file, the workload can be distributed on several machines.
  - 如果大量客户机试图使用同一台服务器，甚至同一个文件，负载可以被分布到几台机器上。

SUN网络文件系统NFS

- NFS简介
- 通 信
- 进程
- 命名
- 同步
- 缓存和复制
- 容错性
- 安全性

NFS简介

- NFS是Sun Microsystem公司1985年研制的网络FS，它是基于UNIX的，是第一个形成产品的DFS。
- NFS正在努力成为工业标准。
- NFS支持不同类型的系统，每台计算机在系统内安装NFS的客户组件和服务器组件。
- NFS的实现思想、协议、实现都非常有特色

NFS简介

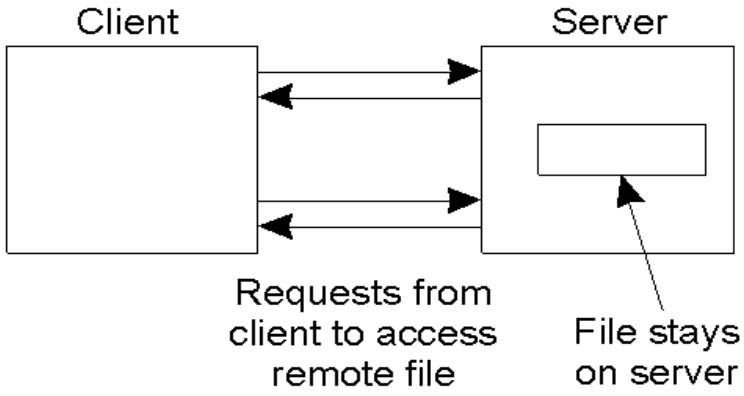
- NFS是Sun Microsystem公司1985年研制的网络FS，它是基于UNIX的，是第一个形成产品的DFS。
- NFS正在努力成为工业标准。
- NFS支持不同类型的系统，每台计算机在系统内安装NFS的客户组件和服务器组件。
- NFS的实现思想、协议、实现都非常有特色

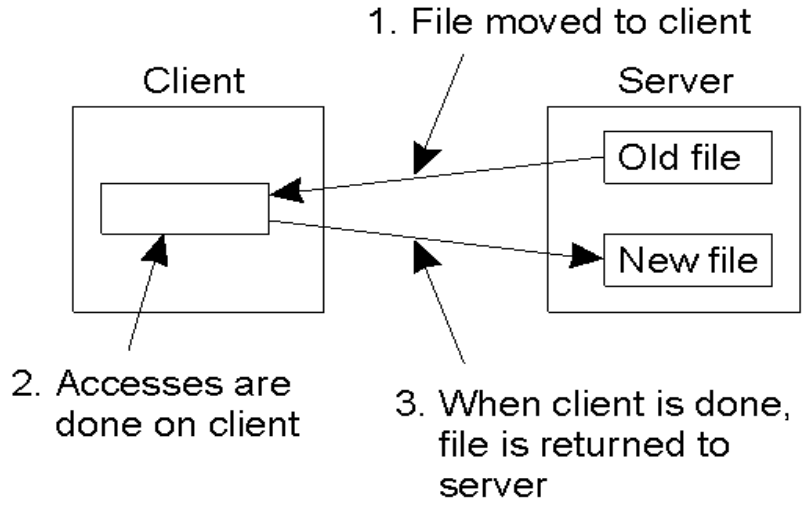
实现思想

- 基本思想：让客户集和服务器的任何一个集合共享一个公用FS，NFS允许一台计算机既是客户机又是服务器。
- 当一个服务器输出某个目录时，该目录为根的子目录树同时被输出。服务器输出的目录列标记在文件的/etc/export中。
- 客户通过安装方式访问服务器的输出目录。
- 基本特征：服务器输出目录，客户从远处安装它们。
- 优点：当多个客户机同时安装同一个目录时，它们可以通过共享公用目录者的文件来进行通信。

NFS 体系结构 (1)

•远程访问模型：客户被提供文件操作的接口，服务器负责实现这些操作。
  •服务器实现的功能较复杂，容易造成服务器瓶颈。
•上传/下载模型：客户从服务器下载文件后，在本地访问该文件；完成对该文件的访问后，再将该文件上传回服务器。
  •服务器功能简单，对文件内部的操作全部由客户端自行完成。所以，要求客户端有较大的空间。
  •另外，每次进行的是整个文件的传送，从而给网络带来许多不必要的压力。

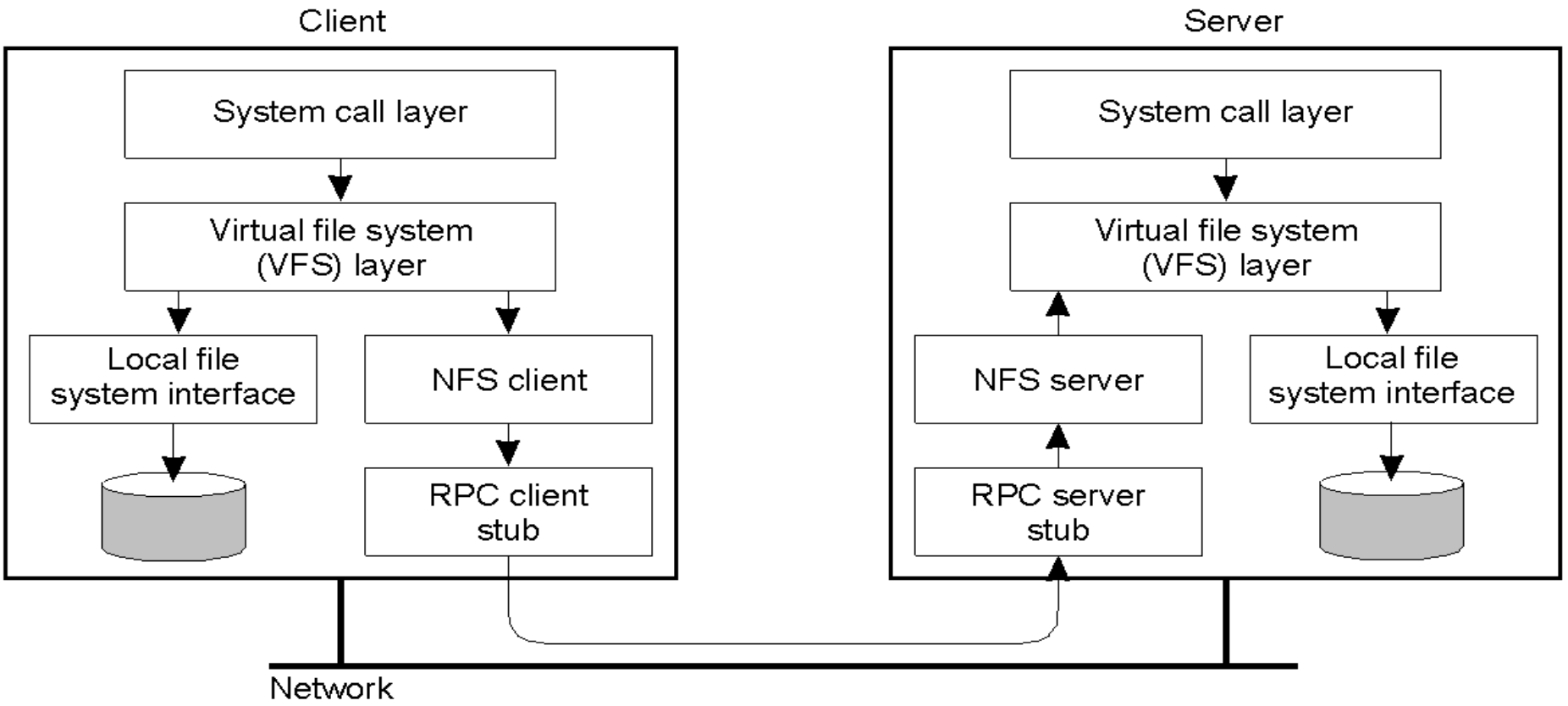




a) 远程访问模型
b) 上传/下载模型

NFS体系结构 (2)

NFS在很大程度上独立于本地文件系统



UNIX系统的基本NFS 结构

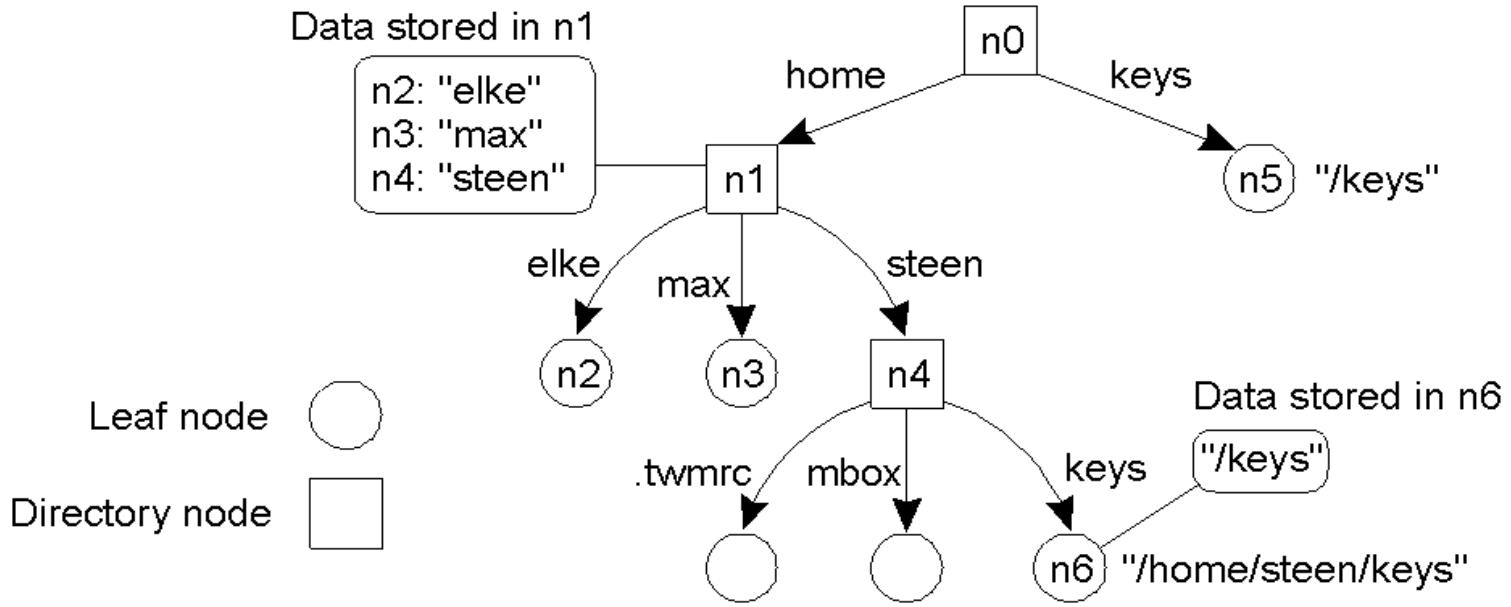
虚拟文件系统 VFS

- VFS对每个文件系统的所有细节进行抽象，使得不同的文件系统在操作系统核心以及系统中运行的其他进程看来，都是相同的。
- VFS并不是一种实际的文件系统。它只存在于内存中，不存在于任何外存空间。
- VFS在系统启动时建立，在系统关闭时消亡。
- VFS拥有关于各种特殊文件系统的公共界面，如超级块、inode、文件操作函数入口等。

文件系统模型

NFS文件系统模型与UNIX系统提供的文件系统模型很类似：

- 文件是字节流，层次化组织在命名图中
- 每个文件有文件句柄
- 支持硬链接和符号链接



文件系统模型

NFS支持的文件系统操作的不完全列表.

Operation	v3	v4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Read the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

SUN网络文件系统

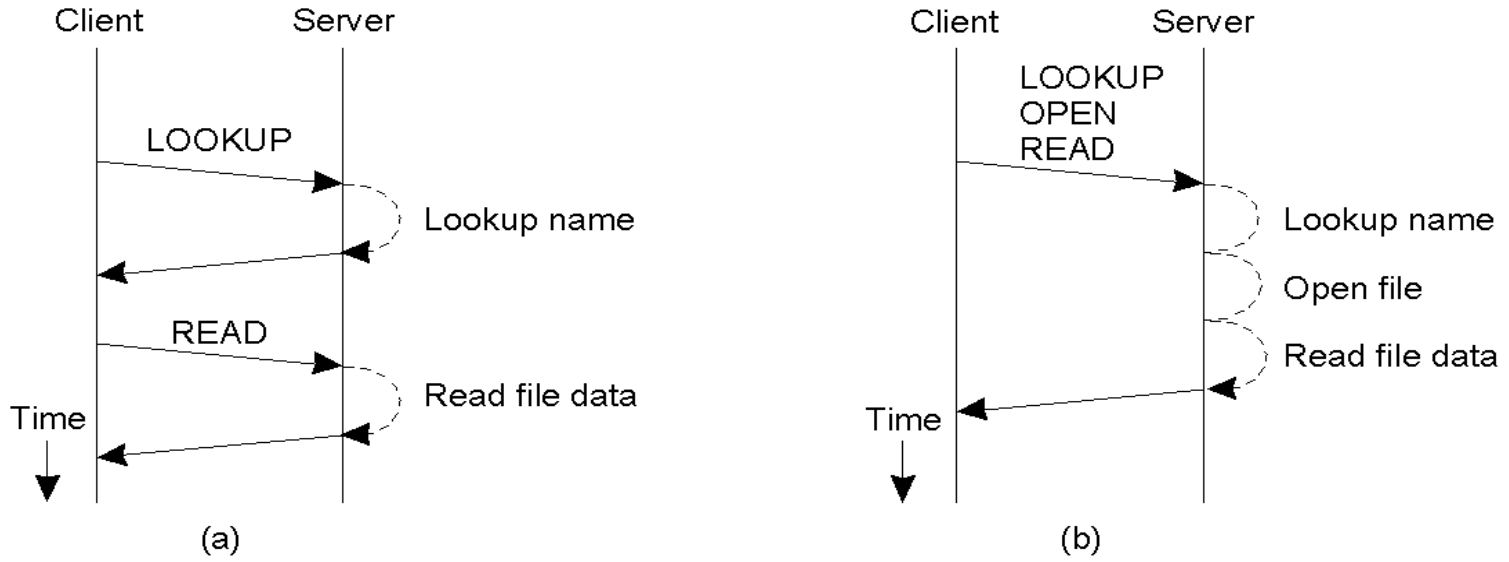
- NFS中的通信

  NFS协议建立于RPC层之上，RPC层隐藏了不同操作系统及不同网络之间的差别。NFS v3一个请求对应一个RPC，而NFS v4多个请求可以对应一个RPC 。

  - 前一种方案中，如果将NFS运行在广域网中，两个以上远程过程调用产生的延迟会导致性能的下降。
  - 后一种方案中，将多个远程过程调用组织到一个单一的请求中，成为一个复合过程。NFS v4的复合过程很明显可以减小延迟并且减少网络中的信息传输。

通 信

每个NFS操作都可以被实现为文件服务器的单一远程过程调用RPC，可以通过支持复合过程来提高性能，复合过程并不包含事务处理语义



a) 在NFS version 3中从文件读取数据

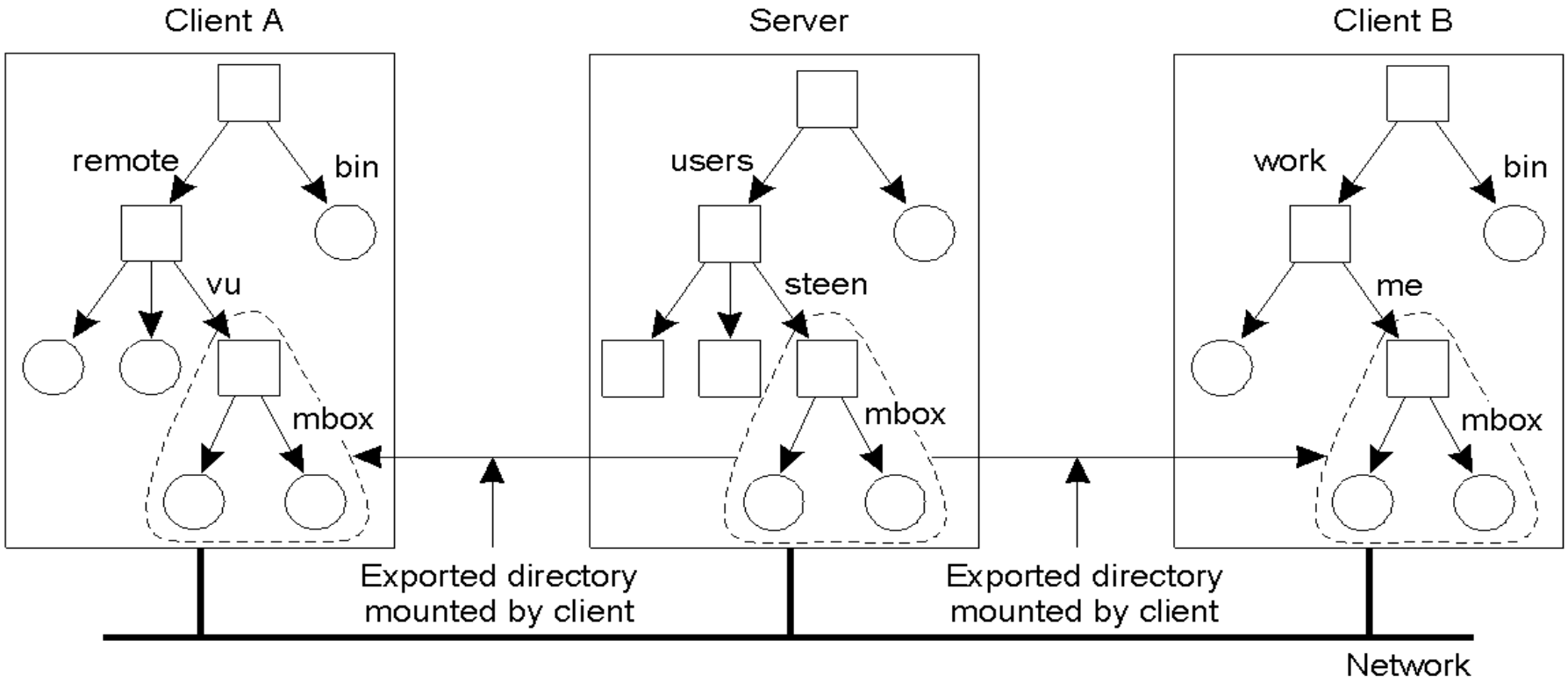
b) 在NFS version 4中使用复合过程从文件读取数据

进程-无状态和有状态的方法

- 在NFS version 3 中NFS协议是无状态的，也就是说，服务器不必保持关于它的客户端的任何协议状态信息
- 优点是简单性：在失败的事件发生的时候，不需要进入回到原先状态的恢复阶段。
- 缺点是客户端得不到任何关于请求是否得以执行的保证。
- 在NFS version 4 中NFS放弃无状态的方法
  - 无状态的模型难以实现文件锁
  - 某些身份验证需要服务器保留客户的状态
  - 需要高速缓存协议，这些协议与保留客户文件信息的服务器合作的最好（如租用）

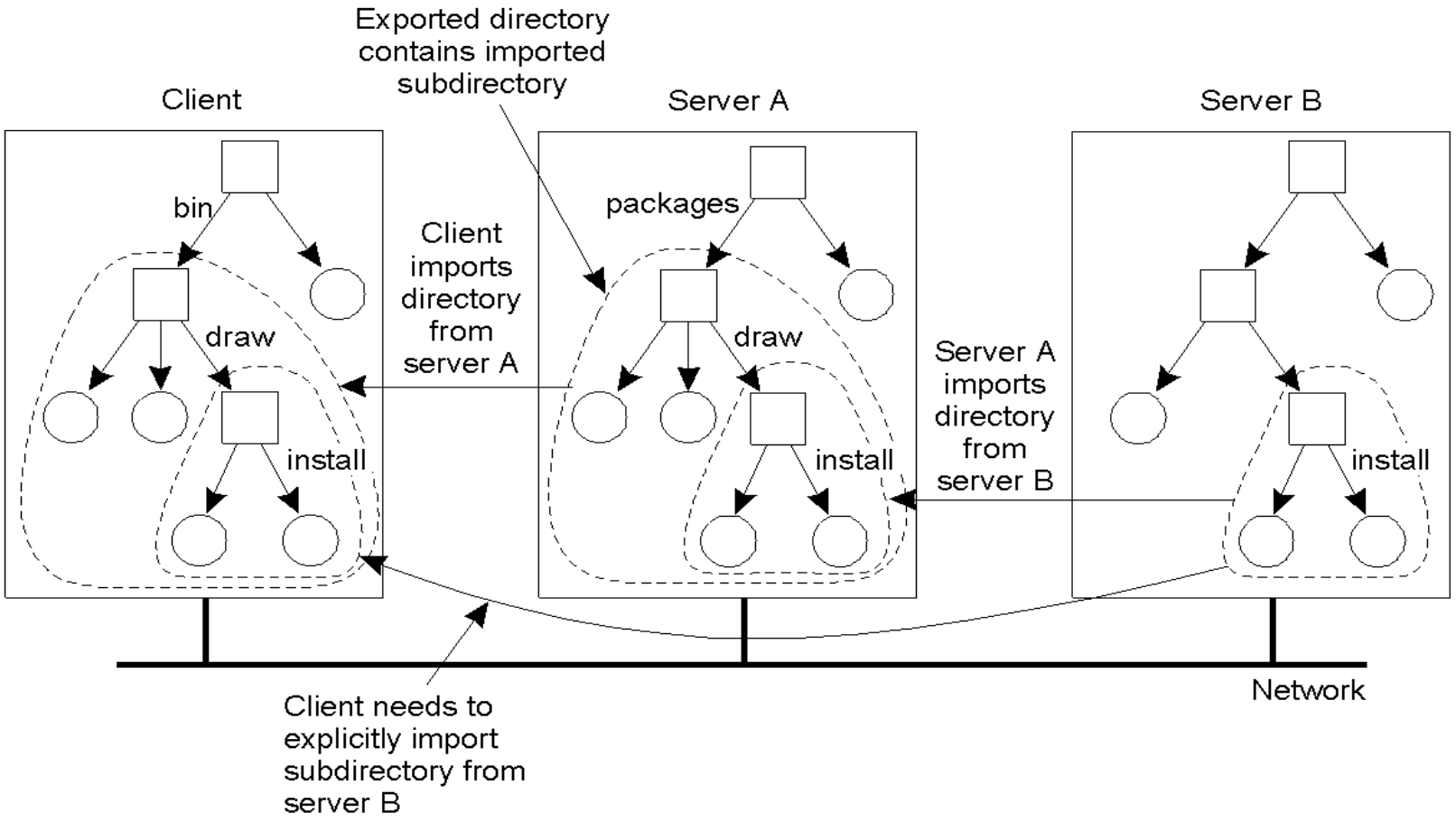
命名 (1)

- NFS命名模型为客户提供完全透明的访问由服务器保存的远程文件系统的机制
- 服务器可以输出（export）某个目录
- 用户不共享名称空间；解决办法：提供一个部分标准化的名称空间



NFS中装入（部分）远程文件系统.

命名 (2)



NFS中从多个服务器装入嵌套目录

命名 (3)

- 文件句柄
  - 对文件系统内文件的引用
  - 对文件是不变的，可以缓存在客户端
- 文件属性

文件属性 (1)

Attribute	Description
TYPE	文件类型 (regular, directory, symbolic link)
SIZE	文件长度
CHANGE	向用户指示文件是否已修改
FSID	服务器上文件所属文件系统的唯一标识

NFS中的强制文件属性

文件属性 (2)

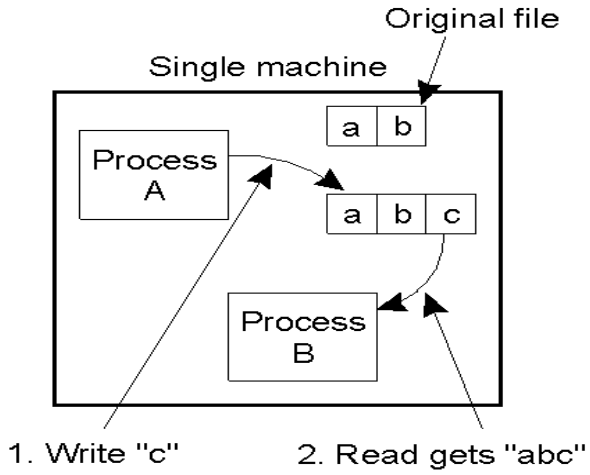
Attribute	Description
ACL	文件关联的访问控制列表
FILEHANDLE	服务器提供的文件句柄
FILEID	文件系统对该文件的唯一标识
FS_LOCATIONS	可能找到该文件系统的网络地址
OWNER	文件所有者的字符串名称
TIME_ACCESS	文件最后访问时间
TIME_MODIFY	文件最后修改时间
TIME_CREATE	文件创建时间

NFS中的推荐文件属性

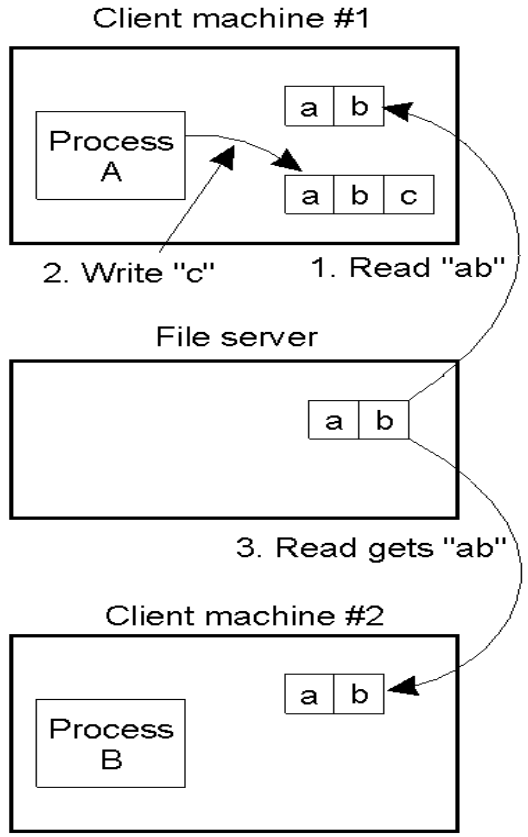
同步-NFS中文件共享的语义 (1)

a) 在单处理器上，**read**操作出现在**write**操作之后时， **read**操作返回的是刚写入的值。

b) 使用缓存的分布式系统可能返回过时的值



(a)



(b)

同步-NFS中文件共享的语义(2)

方法	注释
UNIX 语义	一个文件上的每个操作都对所有的进程都是瞬间可见的
会话语义（**NFS**）	对打开文件的修改只对修改文件的进程（或机器）可见，文件关闭后，所有改动才对其他进程（或机器）可见的
不可改变的文件	不允许更新文件，但允许替换，简化了共享和复制
事务处理	所有改动以原子操作的方式（顺序）发生

四种处理分布式系统中文件共享的方法

同步-NFS中文件共享的语义(3)

不可改变的文件：

- 文件不可更改，文件上的操作只能是**create**和**read**
- 使用新文件替换旧文件：文件不能更新，但目录可以更新
- 替换冲突问题：使用最后一个新文件或随机选择
- 文件被替换，而已被其他进程读取
  - 继续使用旧文件
  - 检测文件已经改变，使随后的读取请求失败

同步-NFS中的文件锁定 (1)

操作	描述
Lock	为一定范围的字节创建锁
Lockt	测试是否已授予冲突的锁
Locku	删除一定范围的字节上的锁
Renew	更新一个指定锁上的租用

NFS version 4 上关于文件锁定的操作

同步-NFS中的文件锁定 (2)

- 锁定文件的隐含方法--NFS共享预约：客户打开文件时，指定它所需的访问类型（READ、WRITE或BOTH）,以及服务器应该拒绝的其他客户的访问类型（ NONE、READ、WRITE或BOTH ）

同步-NFS中的文件锁定(3)

当前文件的拒绝状态

请求访问	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

(a)

请求的文件拒绝状态

当前访问状态	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

(b)

使用NFS共享预约实现操作的结果（新客户试图打开另一客户已成功打开的文件）

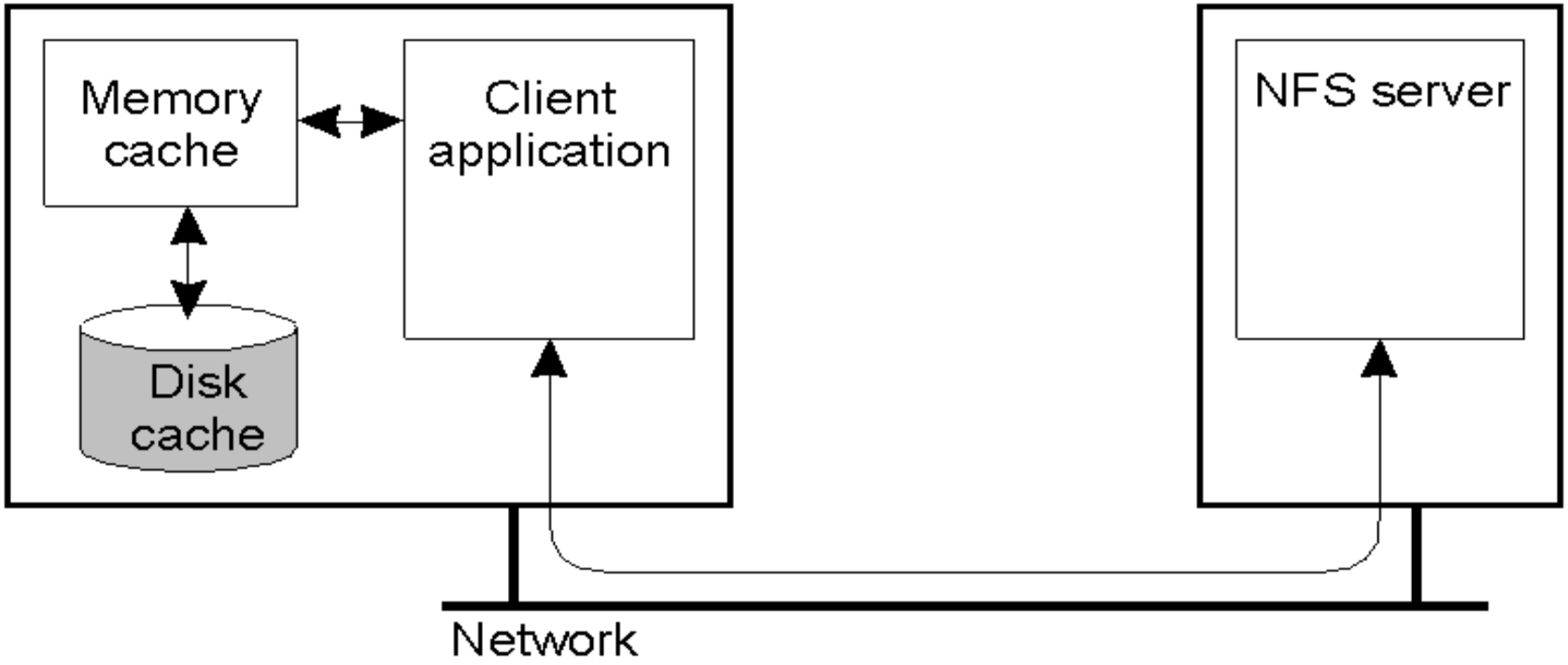
a) 当前文件的拒绝状态下客户请求共享访问的情况

b) 在当前文件访问状态下，客户请求拒绝状态的情况

缓存和复制-客户端缓存 (1)

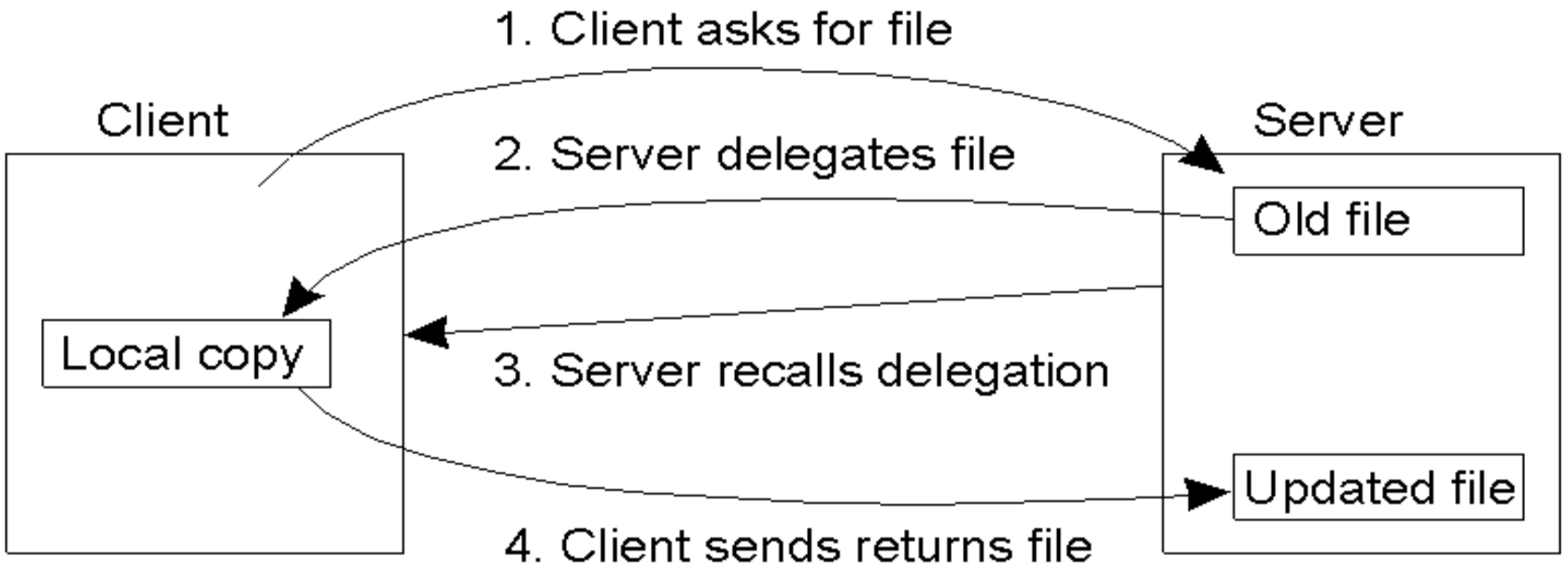
NFS中的客户端缓存：

- 读操作缓存和写操作缓存（实现会话语义）
- 同一机器上的多客户可以共享一个高速缓存



缓存和复制-客户端缓存 (2)

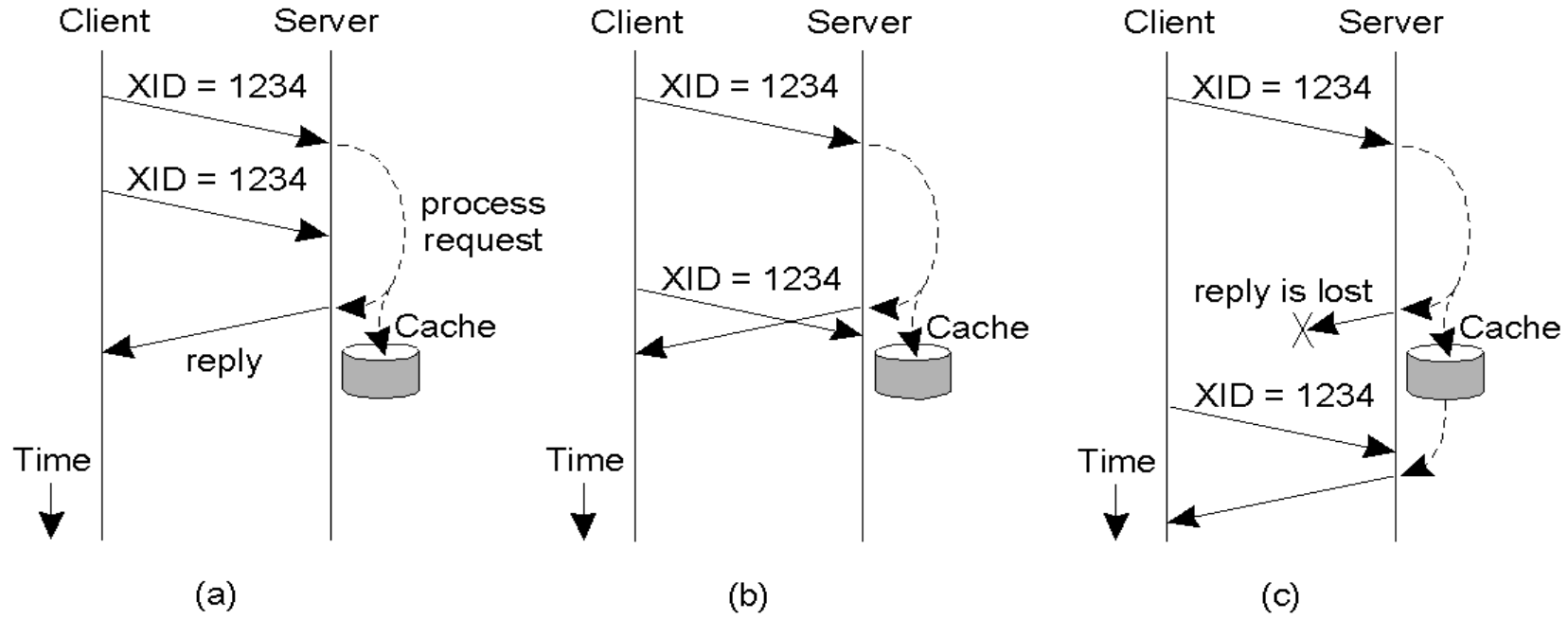
文件委派：服务器可以某些权限委派给用户，如共享预约：
- 来自同一机器上客户的文件锁定可在本地处理
- 服务器拒绝来自其他机器上的锁定请求



使用 **NFS version 4** 回调机制撤销文件委派

容错性-RPC 故障

- NFS的底层RPC不能保证可靠性，而且缺乏对重复请求的检测
- NFS 服务器提供重复请求高速缓存解决：XID事务处理标识符



处理重传的三种情况

a) 请求正在处理

b) 刚返回响应

c) 早已返回响应，但响应丢失

容错性

- 出现故障的文件锁定
  - 客户端崩溃：使用租用
    - 要考虑时钟同步
    - 发送租用更新消息的可靠性
  - 服务器崩溃：进入宽限期，只接收要求归还锁的请求
- 出现故障的打开委派
  - 客户端崩溃：文件修改可能丢失
  - 服务器崩溃：恢复时要求客户返回修改，撤销委派
    - 服务器的文件是最新修改的
    - 服务器再次完全控制了文件，可以将文件委派给其他客户

安全

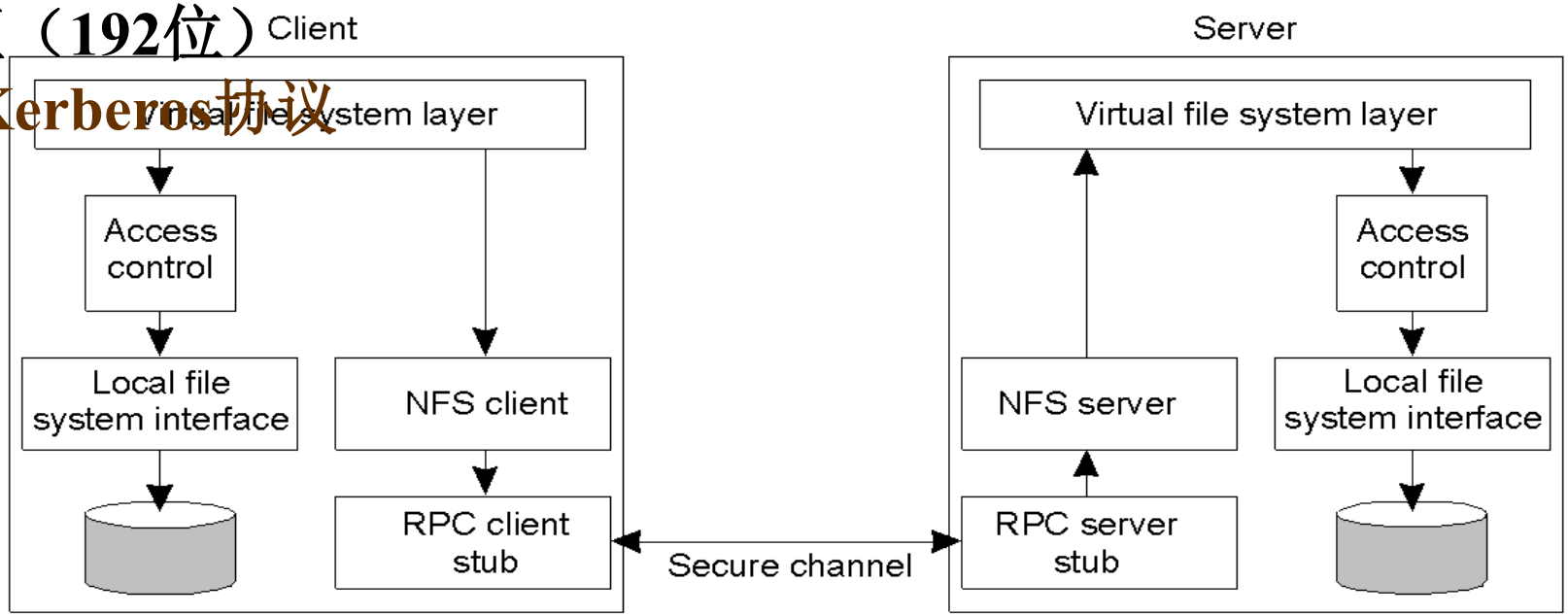
使用安全RPC建立安全通道

在NFS version 4 之前只支持身份验证：

•系统身份验证：客户向服务器以明文发送用户ID、组ID，服务器假设客户已通过登录过程，且信任客户所在的机器

•安全NFS： 使用Diffie-Hellman方法建立共享会话密钥；密钥太短（192位）

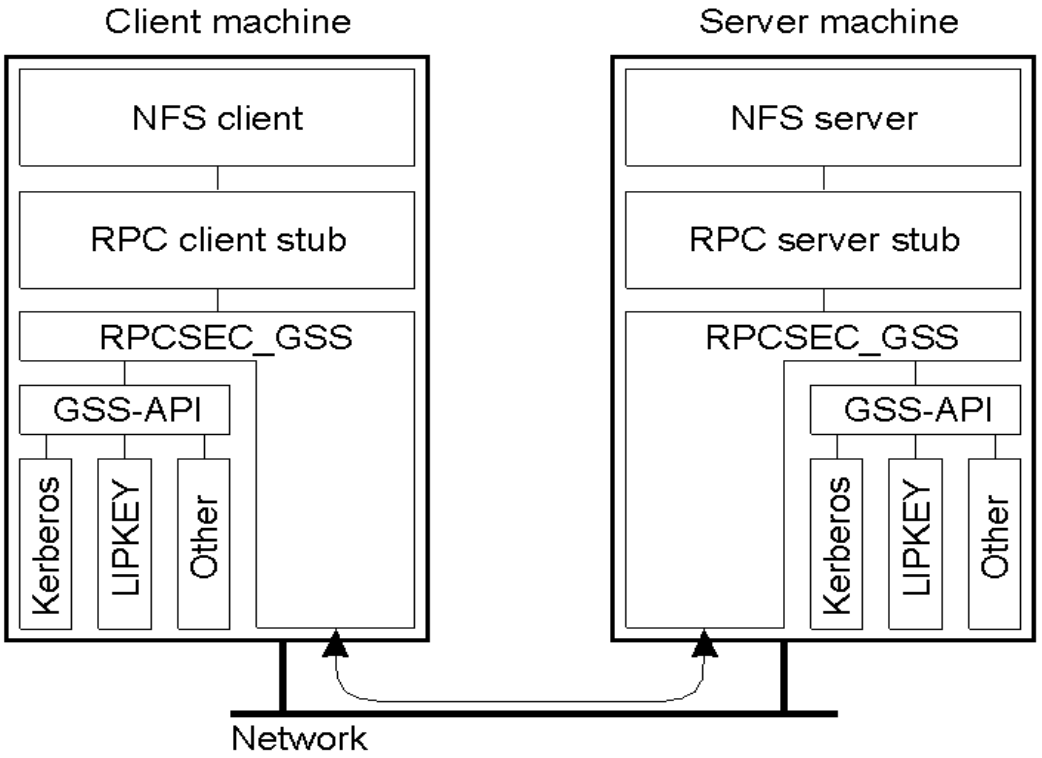
•Kerberos协议



NFS 安全性架构

安全 RPCs

- NFS version 4支持通用的安全框架RPCSEC_GSS
- RPCSEC_GSS不仅支持不同的身份验证系统，而且支持消息的机密性和完整性
- NFS安全设计为不仅支持自己的安全性机制，而且支持处理安全性的标准方法



NFS version 4 中的安全RPC

GSS-API

- GSS-API 提供了三种类型的安全服务：
  - 验证－验证是 GSS-API 提供的基本安全性，它是指对身份进行验证。如果用户通过了验证，则系统会假设其有权以该用户名进行操作。
  - 完整性－完整性是指对数据的有效性进行验证。即使数据来自有效用户，数据本身也可能会损坏或遭到破坏。完整性可确保消息与预期的一样完整（未增减任何内容）。
  - 保密性－保密性可确保拦截了消息的第三方难以阅读消息内容。
- 针对网络应用程序使用 RPC协议的程序员可以使用 RPCSEC_GSS 来提供安全性。
  - RPCSEC_GSS 可提供 GSS-API 的所有功能，但其方式是针对 RPC 进行了调整的。
  - 实际上，RPCSC_GSS 可用于向程序员隐藏 GSS-API 的许多方面，从而使 RPC 安全性具有更强的可访问性和可移植性。

访问控制(1)

NFS访问控制的操作分类

操作	描述
Read_data	Permission to read the data contained in a file
Write_data	Permission to modify a file's data
Append_data	Permission to append data to a file
Execute	Permission to execute a file
List_directory	Permission to list the contents of a directory
Add_file	Permission to add a new file t5o a directory
Add_subdirectory	Permission to create a subdirectory to a directory
Delete	Permission to delete a file
Delete_child	Permission to delete a file or directory within a directory
Read_acl	Permission to read the ACL
Write_acl	Permission to write the ACL
Read_attributes	The ability to read the other basic attributes of a file
Write_attributes	Permission to change the other basic attributes of a file
Read_named_attrs	Permission to read the named attributes of a file
Write_named_attrs	Permission to write the named attributes of a file
Write_owner	Permission to change the owner
Synchronize	Permission to access a file locally at the server with synchronous reads and writes

访问控制 (2)

- NFS中关于访问控制的各种用户和进程

用户类型	描述
Owner	**The owner of a file**
Group	**The group of users associated with a file**
Everyone	**Any user of a process**
Interactive	**Any process accessing the file from an interactive terminal**
Network	**Any process accessing the file via the network**
Dialup	**Any process accessing the file through a dialup connection to the server**
Batch	**Any process accessing the file as part of a batch job**
Anonymous	**Anyone accessing the file without authentication**
Authenticated	**Any authenticated user of a process**
Service	**Any system-defined service process**