

## CS 5523 Lecture 22: Distributed File Systems

- Questions on Laboratories 3 and 4
- Finish Java language security
- Requirements
- Storage system support infrastructure
- How does Unix do it?
- Flat file services
- Directory structure
- Introduction to NFS (Network File System)

---

---

---

---

---

---

---

---

---

---

## Distributed file system requirements:

- Transparency
  - access transparency
  - location transparency
  - mobility transparency
  - performance transparency
  - scaling transparency
- Concurrency control possibility with support for transactions
- Replication
- Fault tolerance
- Heterogeneity
- Security
- Consistency

---

---

---

---

---

---

---

---

---

---

Figure 8.1  
Storage systems and their properties

	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 (Ch. 16)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Persistent distributed object store	✓	✓	✓	✓	PerDis, Khazana

---

---

---

---

---

---

---

---

---

---

Figure 8.2  
File system modules

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

Figure 8.3  
File attribute record structure

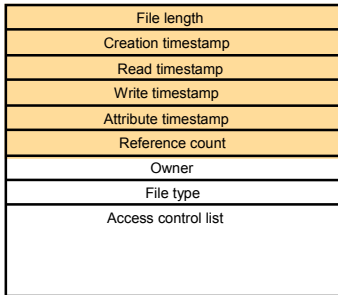
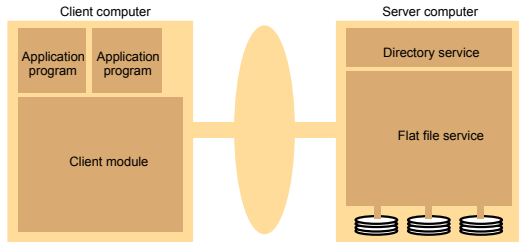


Figure 8.4  
UNIX file system operations

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

Figure 8.5  
File service architecture




---

---

---

---

---

---

---

---

---

---

File service architecture:

- **Flat file service:**
  - provides operations on file content
  - uses a UFID (unique file ID)
- **Directory service**
  - maps text file names to UFIDs
  - may be hierarchical (text names can be directories)
- **Client module**
  - runs on each client
  - emulates local operations
  - may use caching and other strategies for improving performance

---

---

---

---

---

---

---

---

---

---

Figure 8.6  
Flat file service operations

<i>Read(FileId, i, n) -&gt; Data</i> — throws <i>BadPosition</i>	If $1 \leq i \leq \text{Length}(\text{File})$ : Reads a sequence of up to $n$ items from a file starting at item $i$ and returns it in <i>Data</i> .
<i>Write(FileId, i, Data)</i> — throws <i>BadPosition</i>	If $1 \leq i \leq \text{Length}(\text{File}) + 1$ : Writes a sequence of <i>Data</i> to a file, starting at item $i$ , extending the file if necessary.
<i>Create() -&gt; FileId</i>	Creates a new file of length 0 and delivers a UFID for it.
<i>Delete(FileId)</i>	Removes the file from the file store.
<i>GetAttributes(FileId) -&gt; Attr</i>	Returns the file attributes for the file.
<i>SetAttributes(FileId, Attr)</i>	Sets the file attributes (only those attributes that are not shaded in).

---

---

---

---

---

---

---

---

---

---

## Proposed flat file service versus Unix:

- No open or close
- Most operations (except create) are idempotent (UNIX read and write are not)
- Access control is different:
  - ! In UNIX access rights are checked at the open. Process uses a file handle.
  - ! In the proposed service access rights are checked by the server:
    - | Either when a name is converted to UFID access rights are encoded in a capability. The capability is then presented with each subsequent operations
    - | Or, the user credentials are submitted with each request

---

---

---

---

---

---

---

---

Figure 8.7  
Directory service operations

<code>Lookup(Dir, Name) -&gt; FileId</code> — throws <code>NotFound</code>	Locates the text name in the directory and returns the relevant UFID. If <code>Name</code> is not in the directory, throws an exception.
<code>AddName(Dir, Name, File)</code> — throws <code>NameDuplicate</code>	If <code>Name</code> is not in the directory, adds ( <code>Name, File</code> ) to the directory and updates the file's attribute record. If <code>Name</code> is already in the directory: throws an exception.
<code>UnName(Dir, Name)</code> — throws <code>NotFound</code>	If <code>Name</code> is in the directory: the entry containing <code>Name</code> is removed from the directory. If <code>Name</code> is not in the directory: throws an exception.
<code>GetNames(Dir, Pattern) -&gt; NameSeq</code>	Returns all the text names in the directory that match the regular expression <code>Pattern</code> .

---

---

---

---

---

---

---

---

## Proposed directory system versus Unix:

- Text names are translated to UIDs
- UNIX file names are mapped to inodes in a specific device partition
- Hierarchical organization can be supported in both (need a type designation)
- File groups used to allocate blocks of files in a distributed file system to particular servers (analogous to a filesystem in UNIX)

---

---

---

---

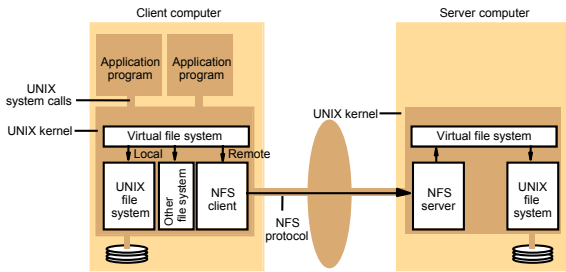
---

---

---

---

Figure 8.8  
NFS architecture



Instructor's Guide for Cindric, Deffense and Kuehling Distributed Systems: Concepts and Design Eds. 3  
© Addison-Wesley Publishers 2000

---

---

---

---

---

---

---

---

---

---

Figure 8.9  
NFS server operations (simplified) – 1

<code>lookup(dirfh, name) -&gt; fh, attr</code>	Returns file handle and attributes for the file <i>name</i> in the directory <i>dirfh</i> .
<code>create(dirfh, name, attr) -&gt; newfh, attr</code>	Creates a new file <i>name</i> in directory <i>dirfh</i> with attributes <i>attr</i> and returns the new file handle and attributes.
<code>remove(dirfh, name) status</code>	Removes file <i>name</i> from directory <i>dirfh</i> .
<code>getattr(fh) -&gt; attr</code>	Returns file attributes of file <i>fh</i> . (Similar to the UNIX <i>stat</i> system call.)
<code>setattr(fh, attr) -&gt; attr</code>	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.
<code>read(fh, offset, count) -&gt; attr, data</code>	Returns up to <i>count</i> bytes of data from a file starting at <i>offset</i> . Also returns the latest attributes of the file.
<code>write(fh, offset, count, data) -&gt; attr</code>	Writes <i>count</i> bytes of data to a file starting at <i>offset</i> . Returns the attributes of the file after the write has taken place.
<code>rename(dirfh, name, todirfh, toname) -&gt; status</code>	Changes the name of file <i>name</i> in directory <i>dirfh</i> to <i>toname</i> in directory <i>to dirfh</i> .
<code>link(newdirfh, newname, dirfh, name) -&gt; status</code>	Creates an entry <i>newname</i> in the directory <i>newdirfh</i> which refers to file <i>name</i> in the directory <i>dirfh</i> .

Continues on next slide ...

Instructor's Guide for Cindric, Deffense and Kuehling Distributed Systems: Concepts and Design Eds. 3  
© Addison-Wesley Publishers 2000

---

---

---

---

---

---

---

---

---

---

Figure 8.9  
NFS server operations (simplified) – 2

<code>symlink(newdirfh, newname, string) -&gt; status</code>	Creates an entry <i>newname</i> in the directory <i>newdirfh</i> of type symbolic link with the value <i>string</i> . The server does not interpret the <i>string</i> but makes a symbolic link file to hold it.
<code>readlink(fh) -&gt; string</code>	Returns the string that is associated with the symbolic link file identified by <i>fh</i> .
<code>mkdir(dirfh, name, attr) -&gt; newfh, attr</code>	Creates a new directory <i>name</i> with attributes <i>attr</i> and returns the new file handle and attributes.
<code>rmdir(dirfh, name) -&gt; status</code>	Removes the empty directory <i>name</i> from the parent directory <i>dirfh</i> . Fails if the directory is not empty.
<code>readdir(dirfh, cookie, count) -&gt; entries</code>	Returns up to <i>count</i> bytes of directory entries from the directory <i>dirfh</i> . Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a <i>cookie</i> . The <i>cookie</i> is used in subsequent <i>readdir</i> calls to start reading from the following entry. If the value of <i>cookie</i> is 0, reads from the first entry in the directory.
<code>stats(fh) -&gt; fsstats</code>	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file <i>fh</i> .

Instructor's Guide for Cindric, Deffense and Kuehling Distributed Systems: Concepts and Design Eds. 3  
© Addison-Wesley Publishers 2000

---

---

---

---

---

---

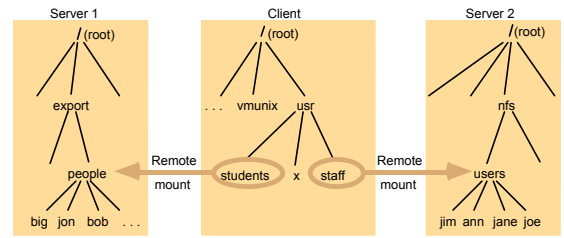
---

---

---

---

Figure 8.10  
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.

Instructor's Guide for Cimbarr, DeRose and Kuehling Distributed Systems: Concepts and Design, Eds. 3  
© Addison-Wesley Publishers 2000

---

---

---

---

---

---

---

---

For next time:

■ Read CDK 8.1-8.4

---

---

---

---

---

---

---

---