Internet File System

Adnan Ilik

Abstract:

This document describes a modular, extendible, and transparent Internet File System (IFS) that provides a common interface for popular Internet services such as FTP.

TABLE OF CONTENTS

1. Motivation 3
2. Overview 3
3. Design 3
4. Implementation 5
   4.1. Redirector 5
   4.2. Dispatcher 5
      4.2.1. File Handles 5
5. FTP File System (FTPFS) 6
   5.1. FTP Overview 6
      5.1.1. Data Connection Overhead 6
      5.1.2. Lack of Parallel or Interleaved Operation Support 6
      5.1.3. Lack of Partial Write Support 6
   5.2. Implementation 7
      5.2.1. ftp_getattr 7
      5.2.2. ftp_readdir 7
      5.2.3. ftp_read 7
   5.3. Optimizations 7
      5.3.1. DNS Cache 7
      5.3.2. Connection-Reuse 7
6. Existing Work 8
7. Conclusion 10

This document describes a modular, extendible, and transparent Internet File System (IFS) that provides a common interface for popular Internet services such as FTP.

Section 1 and 2 explains the motivation behind and the function of the IFS. Section 3 and 4 provides design and implementation goals and overviews. Section 5 discusses the implementation and weaknesses of the FTP file system as a case study. Section 6 talks about existing approaches to the Internet integration problem and their strengths and weaknesses. Section 7 discusses the goals achieved by the IFS.

1. Motivation

On UNIX based systems, the file system provides a global name space for all applications. This name space includes local directories and files, devices (/dev/…), virtual files (/proc/…) that provide information about the machine and the operating system, and remote file systems such as NFS or SMBFS. Existing systems do not allow accessing Internet services through this global name space, thus requiring specialized applications (web browser, FTP client, and news reader) to access data.

2. Overview

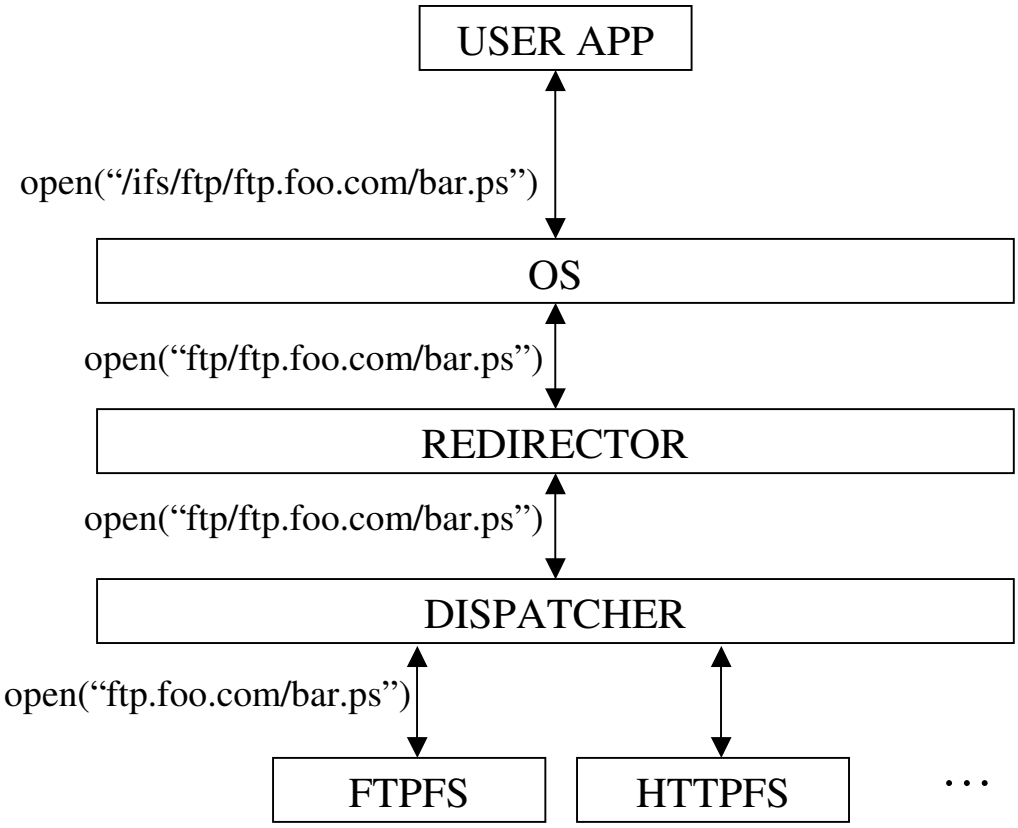
The Internet File System is designed to include the mentioned Internet services in the global name space, thus enabling any application to transparently access Internet objects. For example, a user can view a postscript file on an FTP server by simply typing "`ghostview /ftp/ftp.foo.com/bar.ps`", which is definitely more convenient than using an FTP client to download the file, and then using ghostview to view the downloaded file.

3. Design

Modularity and extendibility has been top priority in the design of the Internet File System. Implementing Internet services as independent modules minimizes cross dependencies, and maximizes extendibility. The modular design makes adding new services or modifying an existing service (e.g. upgrading POP module when POP4 comes out) an easy task.

The IFS consists of 3 layers: redirector, dispatcher, and file system modules. The redirector is responsible for communicating with the kernel and doing lowest level operations such as mounting of the IFS root directory. The dispatcher is responsible for maintaining the IFS root directory, and dispatching requests to file system modules as necessary.

For example, when a request is made for `/ifs/ftp/foo.com/bar.ps`, the redirector sends the request to dispatcher, and the dispatcher forwards the request to the FTP file system module.



Although this level of hierarchy may reduce performance when compared to a hardwired kernel component, it helps with modularity and platform independence. When someone wants to port IFS to a new operating system (e.g. Windows or BeOS), the only component that needs to be modified (or rewritten) is the redirector module.

4. Implementation

Initial implementation of the IFS has been done on the UNIX platform. 6.033 NFS loopback server kit is used as the medium between the operating system and the IFS. The reasons behind this choice are platform independence and ease of debugging.

4.1. Redirector

The redirector consists of two components: NFS client in the kernel mode, and NFS server in the user mode. See "6.033 Spring 2000 Lab3: A Simple File System"[1] for a good overview of the NFS protocol and how a local NFS server allows running a file system in the user mode.

4.2. Dispatcher

The dispatcher component of the IFS is responsible for maintaining the root IFS directory and dispatching requests to file system modules (e.g. FTPFS).

4.2.1. File Handles

The dispatcher module transparently translates between redirector and file-system-specific file handles. When a file handle is returned by a file system (e.g. in response to an open request), the dispatcher tags the handle before sending it to the redirector, and when a redirector passes a file handle (e.g. for a read operation), the dispatcher uses this tag to find the file system module that will handle the request.

---

[1] http://www.mit.edu/~6.033/lab/fs.ps

5. FTP File System (FTPFS)

FTP is the most popular way of transferring files over the Internet, which makes it a good case study for testing and analyzing the functionality, strengths, and weaknesses of the IFS infrastructure. FTP was not designed to be a networked file system, and this makes it a good example for demonstrating the difficulties of integrating Internet services into the conventional file system semantics and architectures.

5.1. FTP Overview

FTP is a protocol – defined in RFC 959[1] – designed to transfer files between computers. There are two types of connections used during an FTP session: control and data.

The control connection is used for authentication and configuration purposes, and the data connection is used for the actual transfer of data. Although this design has its advantages such as third party initiated transfers (see RFC 959 for details), it also has very severe limitations.

5.1.1. Data Connection Overhead

The need to establish a new connection for each directory listing or file transfer operation is very expansive in terms of time (due to TCP handshake delay) and resources.

5.1.2. Lack of Parallel or Interleaved Operation Support

FTP only supports one operation at a time; so simultaneously accessing n files on the same server takes 2n (n control and n data) connections. This limitation significantly degrades performance compared to NFS or SMB sessions.

5.1.3. Lack of Partial Write Support

FTP does not allow partial writes to a file, so if we want to write 4096 bytes at offset 1024000 of a file of size 4096000, we need to download the entire file, modify the range between 1024000 and 1028096, and then upload the entire file. Needlessly to say, such an approach would be unacceptable in terms of performance and atomicity, since FTP does not support locking of files or file regions. Due to this limitation, FTPFS is a read-only file system.

---

[1] http://web/rfc/rfc959.txt

5.2. Implementation

FTPFS module provides 3 functions: ftp_getattr, ftp_readdir, and ftp_read.

5.2.1. ftp_getattr

ftp_getattr is responsible for looking up and getting attributes of an FTP object.

5.2.2. ftp_readdir

ftp_readdir is responsible for retrieving the directory list of an FTP directory.

5.2.3. ftp_read

ftp_read is responsible for reading the specified range of an FTP file.

5.3. Optimizations

There are two optimizations that FTPFS uses to improve the performance: DNS cache, and connection-reuse.

5.3.1. DNS Cache

FTPFS caches hostname lookups in order to improve performance and scalability, since name lookup is the only blocking part of the FTPFS architecture (due to the blocking nature of the gethostbyname function).

5.3.2. Connection-Reuse

FTP supports keeping a connection alive with NOP (no operation) signals, and FTPFS tries to keep all FTP sessions alive. Connection-reuse helps reduce the time spent on socket creation, and TCP handshake.

6. Existing Work

Considering the popularity of the Internet, it is not surprising to find existing attempts to come up with a unifying interface between local and remote services. There are two parts of the Internet integration problem:

- Enabling Internet applications to transparently use existing network resources such as network file and printer servers.
- Enabling all existing applications to transparently access Internet resources such as WWW and FTP servers.

Sun Microsystems' WebNFS[1] project does a very good job in solving the first part of the problem by providing a uniform and platform independent interface to network file systems. WebNFS provides a transparent interface to NFS servers by introducing a new class of URLs in the form of "`nfs://server/path`", thus allows all Java programs that make use of the URL abstraction (through the URL class) to access NFS servers without any modification.

There are a couple of partial solutions to the second part of the problem.

GNU Midnight Commander[2], a file manager, does a very good job in providing a virtual file system interface (FTPFS) to FTP servers. However, this interface is not a part of the global name space and is not visible to other applications.

Berkeley's WebFS[3] provides a transparent file system interface to web servers and primarily focuses on performance and cache coherence. For performance reasons, WebFS closely interacts with kernel mode vnode layer and virtual memory manager. This tight integration makes WebFS very platform dependent.

Alex FTP Filesystem[4] provides a transparent file system interface to FTP servers through a loopback NFS server, making it platform dependent. Also, its lack of modularity makes it very difficult to include support for other Internet services such as HTTP.

UFO[5] provides transparent access to the HTTP name space. UFO has the advantage of being implemented entirely at the user-level using the Solaris proc filesystem to intercept filesystem relevant system calls. Thus, installation of UFO is straightforward and does not require root privileges. However, use of the Solaris proc filesystem limits performance and platform independence.

---

[1] http://www.sun.com/software/webnfs/
[2] http://www.gnu.org
[3] http://www.cs.duke.edu/~vahdat/webfs/webfs.html
[4] http://www.ludd.luth.se/~kavli/alex.html
[5] In *Proceedings of the 1997 USENIX Technical Conference*, Anaheim, CA

In the Windows world, there are two different components that try to solve the Internet integration problem: Internet Explorer (IE)[1] and WinINET[2] library.

IE displays FTP directories in the same way that Explorer displays local directories, thus provides a uniform and transparent interface between local and remote file system objects as far as the user is concerned (e.g. the user can copy files from or to an FTP server just like copying a file from and to a local directory).

The second component that Microsoft introduced for Internet integration is the WinINET library. WinINET library, just like WebNFS, expands the namespace available for applications. All applications making use of the WinINET library can provide transparent access to web objects.

One such application is Microsoft Word 2000, which allows the user to open a remote file (e.g. `ftp://foo.com/bar.doc` or `http://foo.com/bar.doc`) just like opening a local file (e.g. `C:\bar.doc`).

Note that applications that do not use the WinINET library do not have transparent access to the extended name space. So WinINET library does not actually extend the global name space, but provides a secondary URL-based name space.

---

[1] http://www.microsoft.com/ie
[2] http://msdn.microsoft.com/

7. Conclusion

The Internet File System (IFS) provides a complete and platform independent solution to the second part of the Internet integration problem: enabling all existing applications to transparently access Internet resources such as FTP servers.

Unlike existing solutions, the Internet File System (IFS) provides a *modular*, *extendible*, *global* and *transparent* interface to Internet services.