卒業論文 1998 年度 (平成 11 年度)

xxfs:
ネットワーク資源の
ファイルシステムによる抽象化に関する研究

慶應義塾大学環境情報学部村井研究室
関 康治
duck@sfc.wide.ad.jp

指導教員

村井 純
徳田 英幸
楠本 博之
中村 修
植原 啓介

概要

本研究では単一のネットワークファイルシステムプロトコルを通じて、新しい機能を持つファイルシステムを開発するための基礎を製作することを目指す。さらに管理者だけでなく一般ユーザ権限しか持たないユーザでも、ネットワーク上のサービスをファイルシステムに導入できるようにする機構を構築する。ファイルシステムを管理するためのデーモンを動作させ、このデーモンに対して新たな機能を持つファイルシステムのモジュールを登録することで動的にファイルシステムを導入する。

この機構を利用し、他ホスト上には一般ユーザ権限しか持たない人がローカルホスト以外のホストに存在するファイル空間をローカルホストのファイル空間に統合可能なシステムを実装した。

これらの実装は Linux 2.0.36 上で行った。ファイルシステム部分の実装は SunOS Release 5.6 Version Generic でも動作確認し、Linux 2.0.36 上で動作するデーモンに対して SunOS 上で動作するファイルシステムのモジュールをを登録して、Linux 上からファイル操作ができることを検証した。なお、このファイルシステムの実装の登録はユーザ権限で行なった。

またこのシステムの性能を測定するために、ファイルの転送実験を行ない、高速なネットワーク上では、実用に足りる速度が確保できることを確認した。更に新しいファイルシステムの実装にかかる労力がどの程度軽減されたかを確認するため、既存のファイルシステムをこのシステム上で実現した場合にプログラムコード量がどの程度少なくなるかを推測し、相当の労力が軽減されることを確認した。

Abstract

This research aims at a production of base to develop filesystem with new function through one network file system protocol. Its structure also helps those who has end user permissyon, not only administrator, to install a service on network. Concretely, daemon administrates and introduces dynamic filesystem with registre of filesystem implementation.

For example, I implemented a system that enables for an end user to integrate file space on other hosts with one on localhost.

This is completed on Linux 2.0.36. The part of filesystem also works on SunOS Release 5.6 Version Generic; registrating implementation of implementation for SunOS to daemon on Linux 2.0.36, it was confirmed that files can be namaged. This was, incidentally, done with end user permission.

For performance test, some experimentations were completed to reconfirm following two availabilities: one is file transferting speed in practical use on the highspeed network, and the other is effect on coding, decrease of such amount, in case existing filesystem is constructed on this system.

目 次

図目次

表 目 次

Chapter 1

序論

1.1 背景

現在の多くのオペレーティングシステムでは情報の記録にファイルシステムを使用している。

ファイルシステムとは、広大な記憶容量をもつ補助記憶装置に、保存するデータを、その記憶装置の容量に対してより細かい単位で管理するための機構である。ユーザは、このファイルシステムを使用して自分がコンピュータ上で作成したデータやプログラムを保存する。

ファイルシステムは、当初ローカルホストに存在する補助記憶装置を操作するためのみを対象として考えられたものであった。しかし、やがてコンピュータがネットワークを介して操作されるようになると、ネットワークを経由してユーザが利用するファイルを操作する要求が生まれた。このような要求に応えて作られたのがネットワークファイルシステム (NFS) である。

NFS を用いると、別のホスト上に存在するファイルをあたかもローカルホストに存在するかのように使用する事ができる。これはユーザが多くのホストを自由に使える環境では有効であり、広く利用されている。

近年、コンピュータ、特に携帯型コンピュータが普及した。このため、携帯型コンピュータを持ち運び、それを使用して作業することで、ユーザは必ずしもコンピュータのある場所に行って作業する必要性が薄れてきた。また、個人でも手軽に入手することができるようになった PC-UNIX の普及などで、ワークステーションの代わりに携帯型コンピュータを利用して自身の作業環境を構築しているユーザも増えている。

携帯型コンピュータを利用して作業する人が増えてきたことで、携帯型コンピュータをより効率良く使用するために、携帯型コンピュータを利用することを前提考えたファイルシステムを作ろうという研究が出てきた。その例が pfs[5] や wwfs[3, 4] などである。これらは、ネットワークを利用した環境で、ファイル空間をユーザがより柔軟に利用できるよう、ファイル空間を拡張したり、接続状態が良くないネットワーク上でもより利用しやすくするために研究されている。これらは、特に携帯型コンピュータを利用した作業環境で利用すると有効である。

現在、上記のようなネットワークを利用したファイルシステムなどが実現されているが、これらの多くは既存のプロトコル部分のみを利用し、その他のユーザ管理部やサーバ・クライアントの管理などは新しいファイルシステムの開発者が実装していた。

しかし、これは新しいファイルシステムの機能以外の部分に多くの労力を割かなくてはならな

い。このような状況を改善するために、本研究では、新しい機能を持つファイルシステムをより容易に開発できるような枠組を作成する。

1.2 用語の定義

本論文中では用語を以下のように定義する。

ファイルシステム

ファイルを利用するための仕組み・機構。ファイルを操作する際には、ファイルシステムを通じて行なう。この機構はカーネル内に組み込まれる場合や、カーネルモジュールとして提供される場合が多い。本研究においては、ファイルシステムをユーザ空間上で提供する。

ファイル空間

ディレクトリでまとめられたファイルのまとまり。基本的にはディレクトリは木構造で構成される。特に本論中では、あるディレクトリから下流のディレクトリとファイルをまとめてファイル空間と呼ぶ。

ネットワークファイルシステム (**NFS**)

ネットワークを利用してファイルを操作するための仕組み・機構。Sun Microsystems inc. が発表した NFS とは別の意で使用している。特に Sun Microsystems inc. が発表した NFS を表す際には Sun NFS と表記する。

1.3 本論文の構成

本論文では、まず現行のファイルシステムやファイルシステムを開発する環境についての問題点を第 2 章で提示し、その解決案を提案する。そして、解決案である xxfs の目的の明確化を第 3 章で行なう。第 4 章で関連研究について述べる。第 5 章では xxfs の実現の可能性と方向性について論じ、それに基づいて、第 6 章で xxfs の設計と実装について述べる。第 7 章では xxfs の性能等について評価する。最後に、第 8 章は本論文の結論と今後についての課題を述べる。

Chapter 2

現状と問題点

この章では、ファイルシステムについての現状と問題点について述べる。

2.1　ファイルシステムの追加

2.1.1　ファイルシステムの現状

ファイルシステムの多くは単純に補助記憶装置に対してデータを読み書きする機能を持つ。しかし、ファイルを管理するということは、そこに存在するデータを単純に読み書きする事だけではない。

例えば、リビジョン管理システムに見られるように、時間的なファイルの変遷を保持しておきたいこともあるし、一度消去したデータを元に戻したいこともある。しかし、このような機能は、現状ではコマンドとして提供されており、実際に使用する際には、リビジョン管理システムではチェックアウト (co) ・チェックイン (ci) などのコマンドを実行する必要がある。

このような操作は、ユーザの利便性のために自動で行なうことが望ましい。上の例では、ファイルの操作が開始された際にチェックアウトを、終了するとチェックインを自動で行なう仕組みを持つファイルシステムが導入されていれば、コマンドを実行する際の繁雑な操作が避けられる。

現在の UNIX システム上では、emacs に RCS/CVS を組み込む例もあるが、これについても一定の手順が必要であることには変わりない。

また、UNIX システム以外のオペレーティングシステムにおいて、このような機能を持つファイルシステムは存在する。しかしこれは OS に密着に関係しているため、残念ながら一般大勢に使用されるには至っていない。

2.1.2　ファイルシステム設計の現状

OS において、ファイルに関するサービスの実装は多くの場合カーネルに組み込まれて提供されている。

例えばディスクよりデータを読み込み、それを解釈してファイルとして出力する機能を持つモジュールはその機能性から言ってユーザ空間で処理するよりはカーネル内部で処理する方針の方が適している。

しかし、ファイルを読み書きするだけでない、付加価値を持ったファイルシステムを開発する場合、統一的で有効な開発手段は確立されていない。

例えば、リビジョン管理システムを備えたファイルシステムを独自に開発することを考える。リビジョン管理は多くの場合、ユーザが個々にどのようなバージョン管理を行なうかの設定を行なうことや、ファイル毎にバージョンをどのように管理するかなどを細かく設定することができるようにしておくべきである。このような操作はカーネル内で行なうべきではない。ユーザ空間で行なう場合と比較して、デバッグが難しいことや、ファイルをどのように管理するかの設定をを読み込むのが難しいなど、問題が多く存在するからである。

上記のような問題を解決する方法の一つとして、Sun NFS プロトコルなどのカーネルとユーザ空間で通信するファイルシステムプロトコルを仲介に利用してファイルシステムを開発する手法がある。しかし、現況ではこのようなファイルシステムは独自にユーザの管理やプロトコルの理解などを行なうため、この部分の実装に必要以上に多くの労力を費やしている。

また、UNIX 互換のシステム上では、mount システムコールは管理者権限を持つプロセスのみが発行できる。このため、ユーザが新たにユーザ空間で動作するファイルリビジョン管理システムを内包したファイルシステムを開発して使用しようとする場合には、管理者に依頼して、該当ファイルシステムが提供するファイル空間をマウントをして貰うか、mount コマンドがユーザの権限でも動くようにしてもらう必要がある。

2.1.3 改善方法

このような問題を改善するためには、新しい機能を持つファイルシステムをユーザの権限によって動的に導入する必要がある。

mount システムコールは管理者のみが利用できるようになっている。このため、管理者権限によらずに新しい機能を持つファイルシステムを導入するためには、mount システムコールを利用できない。このため、特にファイルシステムを導入する際に工夫が必要になる。この問題を回避するためには、mount によらずに新しいファイル空間を追加する手段を提供すれば良い。

2.2 ファイル空間の拡張

近年は、一人のユーザがネットワーク的に離れた場所に複数のアカウントを持つことも珍しくなくなってきた。このような状態では、あるホストを利用している時に別の管理方針をとるホストや、別の管理主体のホストなどに存在するファイルを使用する必要が出てくる。しかし、多くの場合、ユーザは双方のホストの管理者であることは少なく、必要なファイルを Sun NFS 等によって mount することは難しい。

したがってこのような場合、ネットワークを通じて相手ホスト上で当該ファイルを操作する方法と、当該ファイルを ftp(File Transfer Protocol) などを利用してローカルホストが操作できるファイル空間に複製して操作する方法とが考えられる。ファイルを取得した場合は、一通りの操作が終了した後に元のホストに返却する必要がある。

しかし、ユーザが自分の権限で他ホストのファイル空間をローカルホストに結合できる様になっていれば、より便利にファイルの操作ができる。現状ではこのようなファイル空間の拡張は、容

易には行なえない。

2.2.1　ファイル空間に関する現状

あるユーザが複数のホストにファイル空間を保持する状況下で、いずれのホストからでも自分のファイルを全て参照可能とするには、Sun NFS 等の NFS を用いることが有効である。

しかし、Sun NFS はファイルを公開する側が特定ホストからのファイル操作要求を許可するか否かを明示的に設定する必要がある。この設定をするには、通常は管理者権限が必要である。また、ファイル空間への統合を要求する側も mount コマンドを実行するには管理者権限が必要となる。

ここでは Sun NFS を例としたが、一般に他ホストのファイル空間をローカルホストのファイル空間へ統合するには管理者権限が必要である。

このような問題を解決するためには、管理者権限を必要とせずに、操作するファイルがあるホストのファイル空間を作業中のホストのファイル空間に統合するための機構があれば良い。

ところで、作業に必要なファイルは、該当ユーザの権限で操作して良いかどうかを保証するため、以下の項目によって判断する。

- そのファイルがあるホストにアカウントを持つかどうか
- そのアカウントが作業に必要なファイルに対して操作する権限があるかどうか

したがって、現在使用中のホストのアカウントと、作業に必要なファイルが存在するホストの双方にアカウントが必要となる。

2.3　xxfs の提案

以上の問題を解決する方法として、本研究では新たな機能を持つファイルシステムを容易に開発し、使用できるようにするためのシステムを設計・実装する。

本システムでは、ファイルシステムの開発者に対し、NFS のプロトコルスタック部分を提供することで、開発者はファイル操作部のみを開発すればよいようになる。

また、一般ユーザ権限しか持たぬ者であっても新たにファイルシステムを開発し使用できるため、別のホストに存在するファイルを操作する際にも、ユーザが必要に応じて自分のファイル空間に、必要なファイルを含むファイル空間を組み込むことができる。

このシステムを xxfs(XX File System) [1] と呼ぶ。

---

[1] xxfs は x を未知の文字と解釈し、「どのようなファイルシステムでも導入できる」という意味を持たせている、XX File System と呼んでも良いし、wwfs(World Wide File System もしくは Waku Waku File System) の次世代ファイルシステムという意味を持たせている、w+1=X w+1=X File System と呼んでも構わない。

Chapter 3

xxfs の目的

本章では、xxfs の目的について述べる。

3.1 目的

Chapter 2 で述べた問題点に基づいて xxfs の実現に向け研究を行なう。
本研究においては下記の項目を実現する事を目的とする。

- ユーザ権限しか持たないものが新たなファイルシステムを導入可能にする。
- 一般ユーザ権限しか持たない者がその権限の範囲内でファイル空間の拡張可能にする。
- ファイルシステムの開発者が容易に新たなファイルシステムを設計・導入可能にする。

携帯型などの可搬性のあるコンピュータを持ち運びつつ使用する、移動体コンピューティングを行なう上では、どのようなネットワークに接続されていても同じような環境で作業できるということが重要である。しかし、ネットワーク毎の管理方針の違いやネットワーク・ホストの構成などによって、必ずしもどこでも同様な環境が用意できるとは限らない。

また、複数のホストを使用することが可能な者にとって、どのホストを使用する際にも同じような環境を実現するためには幾つかのアプローチがあるが、本研究ではファイル環境に着目して解決を図る。コンピュータにおけるユーザの作業は、一般にアプリケーションソフトを利用して結果をファイルに反映させるという操作に抽象化できる。

ユーザが日常的に使用しているホスト以外のホストにおいて、日常的に使用しているホストと同様に作業しようとする場合を考える。このとき、日常的に使用しているものと同一のアプリケーションソフトがあることと、操作するファイル空間が同一であることで、日常的に使用しているホストと同様の環境で作業できるようになる。

ファイル環境という言葉にも幾つかの段階があるが、本研究ではファイル空間の拡張とファイルシステムの拡張という視点で考える。

3.2 ファイルシステムの拡張

ユーザ権限しか持たぬユーザが、ファイルシステムを容易に導入できるようにするための機構を考える。

現在、新たな機能を持つファイルシステムを導入・構築するためには、管理者の権限が必要である。しかし、ファイルシステムを利用するのはあくまでユーザであり、管理者の想定するホストの使用形態が必ずしもユーザに使いやすいものであるとは限らない。ユーザが自分に与えられた権限を最大限有効に活用すれば自分が必要とするファイルシステムを拡張できるものを、該当する技術が無いために実現できないということは、ユーザにとって利益を損ずる。

xxfs では、管理者権限を必要とせず、新しいファイルシステムを導入できる仕組みを提供する。

しかし、xxfs 自身も現況では新しいファイルシステムであるので、これを導入するためには管理者権限が必要である。

xxfs の導入には管理者権限が必要であるが、xxfs 上で稼働する新しいファイルシステムの導入には管理者権限は必要でない仕組みを提供する。

3.3 ファイルシステムの実装

ファイルシステムの開発者の立場では、新しいファイルシステムを開発するために多くの考えなくてはならない要素がある。本来目的となる機能の他に、ユーザ認証の問題やセキュリティ, ホストの解決, 通信方法など、ファイルシステムの機能の本質的でない部分の開発のために多くの時間を割く必要がある。このため新しいファイルシステムが実装されず、ユーザが享受できるはずの利益が得られないということは損害である。

このような状況に陥らないようにするために、より容易に設計・実装・導入できるファイルシステムを製作するべきである。

3.4 ファイル空間の拡張

ユーザが独断で自分が使用できるファイル空間の拡張をするための手段を考える。

ファイル空間を拡張するためには mount システムコールを用いるので、ユーザが自分自身でファイル空間を拡張するのは難しい。

3.4.1 Sun NFS によるファイル空間の拡張

ユーザが使用するファイル空間をユーザ自身が拡張可能にすることで、より使いやすい環境を作り上げる。複数のコンピュータにおいて同様の操作感を出すために、ホームディレクトリとユーザ情報の共有をする環境を実現することを目的として Sun NFS と Sun NIS を用いることが多い。Sun NFS はユーザ情報を共有していることが前提になっているため、Sun NIS 等によってユーザ情報が同一の環境に限って有効に動作する。

また、mount 要求側に対する認証も必要であるが、この設定権限は管理者に属し、一般ユーザにはない。

3.4.2 NFS とデーモンを利用したファイル空間の拡張

ユーザ権限によるファイル空間の拡張を実現する手法の一つに、NFS を用いる方法がある。あらかじめ NFS クライアントより mount してあるネットワークデーモンが、ファイル操作命令を適宜判断して別の操作とすることで疑似的にファイル空間を拡張する。

この手法をとったファイルシステムの一つに wwfs がある。wwfs のシステムは NFS クライアントからのファイル操作命令を解釈し、その命令に応じて FTP を利用してファイルの取得や操作を行なう。この手法をとることで、FTP で提供されているファイル空間をユーザが使用できるファイル空間に統合することができるようになる。

3.4.3 xxfs におけるファイル空間の拡張

本研究ではユーザ権限しか持たない者が、普段使用しているホストに管理権限を持たない場合であっても、任意の端末においてホームディレクトリをファイル空間に追加できるような機構を提案する。

これは、wwfs と似た手法を用い、ネットワークデーモンと NFS プロトコルを使用して、ネットワークデーモンで仮想的にファイル空間を拡張するという方法で行なう。

Chapter 4

関連研究

ここでは、xxfs のシステムを実現する上で関係のあるシステムについて見て行く。

4.1 Sun NFS

Sun NFS は他のホストに存在するファイル空間を、あたかもローカルホストのファイル空間の一部であるかのように操作できるファイルシステムである。

4.1.1 特徴

Sun NFS の特徴は、使用者から見て同じ環境を容易に準備可能であるということである。

これ以前のシステムにおけるファイル空間はホストに存在するもののみを参照可能であった。普段使用していないホストを使用する人間が、普段使用しているホストにあるファイルを使用するためには、UUCP や FTP などのファイル転送を行ない、ローカルホストにファイルを取得する手法か、普段使用しているホストにネットワークから login して使用する方法を利用していた。

しかし、Sun NFS と NIS を併せて利用することで、どのホストを利用してもユーザはホームディレクトリを共通に使用できる。これによって、どのホストを利用したとしてもほぼ同様の作業環境を得ることができるようになった。

4.1.2 認証

mount

ホームディレクトリや共通に使用するバイナリファイルをマウントしてファイル空間を拡張するためには、管理者の権限が必要である。

mount 命令は、ファイル空間のセキュリティ確保のため、管理者のみしか実行することができない。Sun NFS はカーネルから直接 mount するため、管理者のみがファイル空間の拡張ができる。このため、ユーザは自分のファイル空間を自分自身で拡張することはできない。

export

Sun NFS のファイル空間をマウントされる側についても、export の許可が必要になる。

具体的には /etc/exports ファイルにファイル空間の提供を許可するホストとファイルツリーの対を記述しておくことで、export の許可認証をしている。もしみだりに export を許可すると、望まない相手にもファイル空間を mount されてしまい、大切なファイルを読まれてしまう可能性や、数々のセキュリティホールになる可能性があるためである。

ユーザ管理

Sun NFS をユーザのホームディレクトリなどとして使用する際には、NIS(Network Infomation Service) などによってそれぞれのホストで同じ UID(User ID)、GID(Group ID) の設定をしておく必要がある。Sun NFS では UID や GID は元のホストに存在する情報をそのまま mount した側のホストに流すためである。例えば、UID 100 番のユーザが元のホストでは A というユーザが使用しているが、mount した側のホストでは B というユーザが同 UID を使用していた場合には、ユーザ B が元のホストのユーザ A のファイルを読み書きできてしまうことが考えられる。このため、例えばホームディレクトリとして Sun NFS を使用するユーザはどのホストで使用する際にも同じ UID, GID を使用する必要がある。複数のホストで同じ UID, GID を使用させるための仕組みとして NIS が使用されることが多い。NIS はクライアント・サーバ方式により一つのホストでユーザ情報を集中的に管理する手法を取っている。サーバとなるホストでユーザ情報を登録, 変更することで、クライアントとして動作しているホストではそれぞれ相互に同一の UID,GID を使用することが出来る。

4.1.3 ファイルシステム拡張の手段としての Sun NFS プロトコル

Sun NFS が持つ、別ホストのファイル空間を扱えるという特徴を利用して、新しいファイルシステムの実装に Sun NFS を使用している場合がある。

特にネットワーク上のサービスや、ネットワーク越しのファイルツリーを何らかの手段でマウントする場合には Sun NFS を使用することが多い。例えば wwfs や pfs はその典型である。

Sun NFS プロトコルは RPC を使用して作られている。RPC を使用することで、関数呼び出しの形でファイルを操作する関数群を実装すれば新たなファイルシステムを作成できることができる。このため、新しいファイルシステムを比較的容易に開発することができる。しかし、Sun NFS プロトコルにおいてはユーザ認証については何も保証していない。これを保証するためには、新しいファイルシステムにおいてはユーザ権限の確認などは開発者が考える必要がある。

4.1.4 xxfs との比較

xxfs が目的とする機能と Sun NFS が提供している機能との比較を表 4.1 に示す。

4.2 wwfs

4.2.1 特徴

wwfs は Sun NFS を使用して FTP で提供されるファイルシステムをローカルホストのファイル空間に統合するためのシステムである。

表 4.1: Sun NFS と xxfs との比較

	Sun NFS	xxfs
ファイル空間の拡張	双方のホストに管理者権限が必要	xxfs の準備があればユーザ権限のみで可
速度	高速に操作可能	構造的に Sun NFS よりは遅い
システムの堅牢さ	ユーザ情報が閉じていれば充分に堅牢	他のホストに情報を提供できる分堅牢でない
柔軟性	ファイルの提供を目的としている	より多目的に使用できる

これにより、管理者権限を必要とせずにユーザのファイル空間を拡張することができるようになる。

4.2.2 認証

ユーザ認証

wwfs におけるユーザ認証はローカルホストにあらかじめ設定してある設定ファイルに認証情報を書き込んでおくことで行なっている。設定ファイルの中にユーザ名とパスワードを、使用者が書いておき、そのファイルを wwfs が FTP での接続時に参照することで、相手ホストの認証に対応する。

マウント

wwfs は Sun NFS をマウントするのと同様にマウントできる。wwfs とカーネルとの通信部分に Sun NFS プロトコルをそのまま使用しているため、Sun NFS をマウントする時の記述と同様に管理者権限でマウントし、ユーザはマウントしたディレクトリより下の階層を使うことになる。

4.3 pfs

4.3.1 特徴

pfs は高遅延ネットワークにおけるファイル空間の使用を考えたファイルシステムである。pfs クライアントサイドデーモンの中でファイルをキャッシュとして保持し、高遅延ネットワーク使用せざるを得ない場合やネットワーク接続がされていない場合には、キャッシュされたファイルを参照する。このため、高遅延のネットワークを通らずにファイルの操作ができるので、いちいちネットワークを通じてファイル操作要求を送受信するのに比べ高速にファイルの操作が可能である。

4.3.2 構成

pfs はファイルのキャッシュ等を保持するクライアントサイドデーモンとなる部分と、そのデーモンにファイルを供給するファイルサーバとなる部分、マスターサイドデーモンで構成される。

デーモンの部分とファイルサーバ部分は pfs プロトコルで通信しており、pfs は通信状態を判断して、それに応じて内部のキャッシュを利用するかファイルを取得するかを決断している。

4.4 xxfs との比較

pfs と xxfs の比較を 表 4.2 に示す。

表 4.2: pfs と xxfs との比較

	pfs	xxfs
回線接続時のファイル操作	ファイルキャッシュにより高速	現状では構造的に pfs より遅い
回線切断時のファイル操作	ファイル操作により可能	不可能
提供の目的	リモートホストにあるファイルをローカルホストに提供することにある	新規ファイルシステムの拡張が目的である

pfs と xxfs は機構として似通った形態を取っている。しかし、pfs はその主な用途が高遅延ネットワーク、もしくは不安定なネットワーク環境におけるファイル操作を目的としたものであるが、xxfs は新しい機能を持つファイルシステムを容易に開発し、導入できるようにすることを目的としたものである部分が異なる。

Chapter 5

xxfs 実現へのアプローチ

本章では xxfs システムの実現の可能性について述べる。

5.1 問題点の整理

ユーザが新しいファイルシステムを導入できるようにするためには幾つかの特殊な構造が必要となる。ファイルシステムを新たに導入するためには、以下の問題がある。

- ファイルシステムモジュールの導入
- ファイル空間への追加

上記問題点について、解決法を述べる。

また、xxfs を導入することで、同時に実現できるようになる、管理者権限がなくともファイル空間を統合できる機能についての実現法について述べる。

5.2 ファイルシステムモジュールの導入

ファイルシステムモジュールは、従来はカーネルによるサービス、もしくはカーネルモジュールによるサービスによって実現される。しかし、これらではファイルシステムモジュールを新規に導入するにあたり、カーネルの再構築やカーネルモジュールの導入などシステムに変更が必要となる。また、これらの操作はホスト管理者のみができるものである。

ファイルシステムの新規導入を容易にするために、xxfs の基本となる部分には既存のファイルシステムの一部を利用する。具体的にはネットワークファイルシステム (NFS) のインタフェース部分を流用することで実現する。ファイル操作の要求をカーネルから外部のプロセスに伝えるためには、インタフェースとしての役割を持つ部分が必要である。NFS のうち、カーネルに実装された NFS クライアント部分がこの役割を担当する。

NFS は NFS プロトコル を用いて、別のプロセスとして動く NFS サーバ にファイル操作の要求を行なう。この NFS サーバ 部分を新しいファイルシステムとしての機能を持つプロセスに変更することで、単純なファイル操作以外の機能を持たせることができる。また、ネットワークプ

ロトコルはユーザ権限によらず使用することができるため、このプロセス自体はユーザ権限で動作させせることができる。

5.3　ファイル空間への追加

ユーザが新しいファイルシステムモジュールを導入することはできても、それをファイル空間に追加することはできない。ファイル空間に別のファイル空間を追加するためには、mount システムコールを発行する必要あるからである。mount システムコールを実行するには、管理者権限が必要である。

このため、mount システムコールによらず、新しいファイルシステムをファイル空間に統合するための仕組みが必要である。この仕組みを図 5.1 に示す。

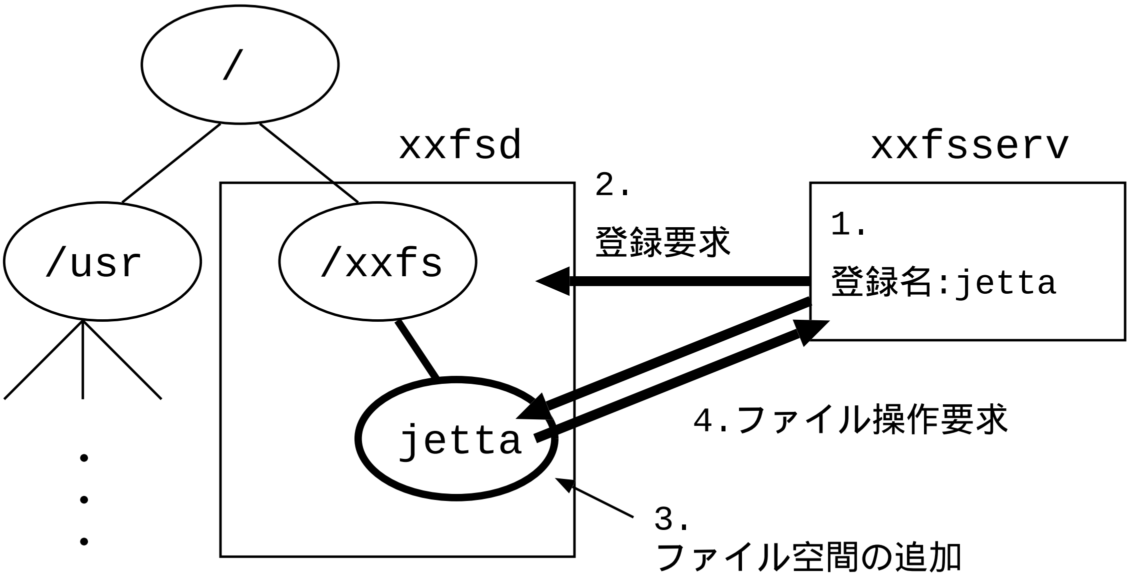

図 5.1: ファイル空間への追加

本研究では、この仕組みを実現するために別のネットワークデーモンを用意した。このネットワークデーモンは、ホストからの mount システムコールによりファイル空間ディレクトリとして統合される。このネットワークデーモンはファイルシステムモジュールを管理する機構を持ち、NFS クライアントからのファイル操作要求に応じて適切なファイルシステムモジュールにファイル操作要求を送り直す。本論中では、このネットワークデーモンを xxfsd と呼ぶ。

xxfsd に、新たなファイルシステムモジュールを登録し、ファイル操作要求をそのファイルシステムモジュールに転送するよう設定することで、xxfsd によって提供されているファイル空間に新たなファイルシステムが提供するファイル空間が追加される。追加されたファイル空間は、実際には新たなファイルシステムモジュールによって操作される。このようにしてファイルシステムの導入を行なうことができるようにする。本論中では、このファイルシステムモジュールを xxfs サーバ と呼ぶ。

xxfs サーバ は、ユーザ権限でも動作可能とする。

5.4　ファイル空間の拡張

ユーザ権限でファイル空間を拡張するためには、現状ではファイルシステムモジュールの導入と同様に管理者権限が必要となる。しかし、予め xxfsd を動作させておき、xxfsd に対してファイルを実際に操作するプロセスである xxfs サーバ を登録することで、mount システムコールを利用しなくてもファイル空間を追加することができる。

実際の動作は「5.3 ファイル空間への追加」で論じた方法と同様の手法を取る。なお、別ホストに存在するファイル空間を使用しているホストのファイル空間に追加するだけならば、xxfs サーバ の機能は通常のファイル操作要求に沿ったファイル操作を忠実に実行するものとなる。

Chapter 6

xxfs の設計と実装

本章では xxfs の設計について説明する。

6.1　システムの概要

xxfs システムの概要を図 6.1に示す.

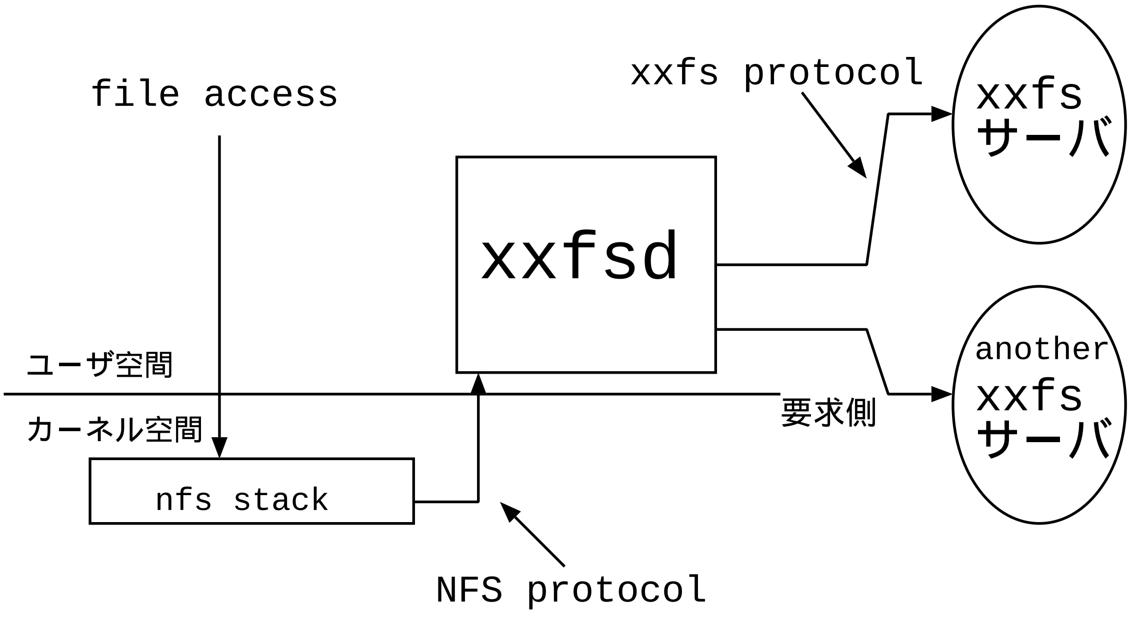

図 6.1: xxfs の概要

xxfs は xxfsd と xxfs サーバ から構成される。

ユーザがファイル操作要求を行なうと、そのホストのカーネルの Sun NFS スタックを通じて xxfsd へファイル操作要求を転送する。xxfsd はその操作要求を解釈し、内容によって、xxfs サーバ に転送する。xxfs サーバ はファイル操作要求に応じて、実際のファイル操作を行ない、その結果を xxfsd 経由でカーネルの Sun NFS スタックへ返送する。

xxfs はユーザ空間で動作する。これはカーネル空間上に実装した場合には、以下のことが問題になるためである。

- デバッグの難しさ
- システムとしての設定の難しさ
- 移植性のなさ
- ローカルホスト以外では xxfs の恩恵を受けることができない

xxfsd がデーモンとして動作することで、xxfsd が実行されているホスト以外のホストでも xxfs が供給する機能を享受することができる。

xxfsd

xxfsd はカーネルからの要求を解釈し、適切な xxfs サーバ にその要求を転送する役割を持つ。

xxfsd はカーネルと Sun NFS プロトコルを用いたネットワーク通信を。また、xxfsd は xxfs サーバ と xxfs プロトコル を用いたネットワーク通信をする。

xxfsd は内部に自身が管理している xxfs サーバ のリストを持つ。xxfsd が カーネル からファイル操作の要求を受け取ると、その要求に応じて適切な xxfs サーバ を判断し、その xxfs サーバ に要求を転送する。

xxfs サーバ のリストは xxfs サーバ 起動時に xxfs サーバ から xxfsd に送る情報に基づいて作られる。xxfs サーバ から送られる情報は xxfs サーバ が動作しているホスト名・ポート番号・登録名からなる。

xxfs サーバ

xxfs サーバ は一般ユーザ権限でも動かすことができるネットワークファイルサーバである。xxfs プロトコルを解釈して、xxfsd からのファイル操作要求を実行する。xxfs サーバ は起動時に xxfsd に登録の要求を送り、その後 xxfsd からのファイル操作要求を待つ。

6.2 xxfsd の構造

xxfsd は大きく以下の部分に分類する事ができる。

- NFS クライアントからのファイル操作要求受付部
- NFS クライアントからのマウント受付部
- xxfs サーバからの登録要求受付部
- xxfs サーバへの要求発信部
- xxfsd のメイン部

これらの構造を、図 6.2に示す。

xxfsd は同時に 3 つの要求待ち受けを行う。マウント要求の待ち受けと NFS クライアントからのファイル操作要求待ち受け、xxfs サーバ の登録要求待ち受けである。

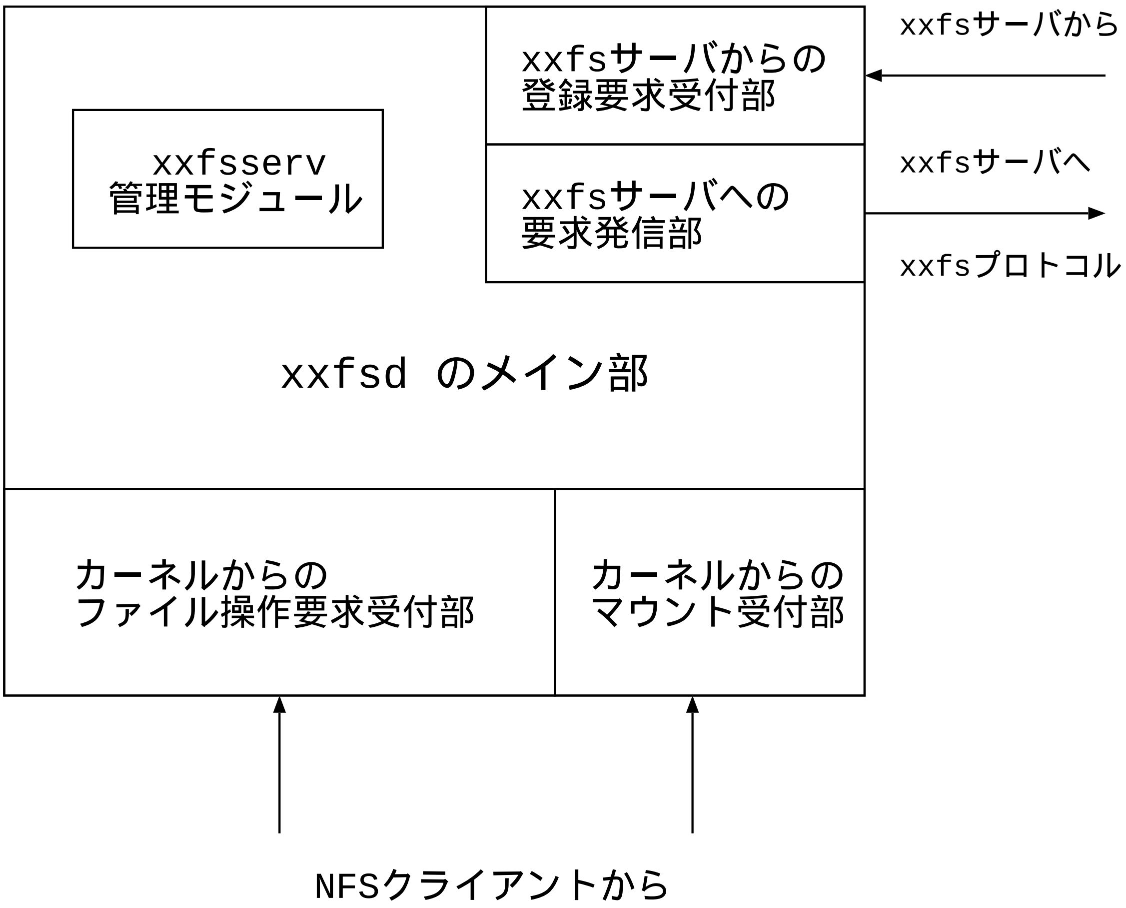

図 6.2: xxfsd の構造

これらを効率良く実現するためには、別の機能を持つ部分をを一つの待ち受けルーチンにまとめる必要がある。これらをまとめることで、それぞれの部分に対して別のルーチンを実装する必要がなくなる。

具体的には、それぞれの待ち受け部を RPC(Remote Procedure Call)[11] を利用して実装することで、効率的な要求待ち受けを実現する。xxfsd は起動と同時にこれら 3 つのプロトコルの待ち受けリクエストを portmapper に登録する。いずれかの待ち受け部にリクエストが来た場合には、portmapper からの情報を元に適切な機能が呼び出される.

6.2.1 ファイル操作要求受付部

ファイル操作要求受付部は、mount した側のホストでユーザがファイル操作を行なった際に発生する、ファイル操作要求を受け付ける部分になる。この部分では、要求の待ち受けを行ない、NFS クライアントより要求があると、受け取ったファイル操作要求を xxfsd のメイン部に転送する役割を持つ。

この部分は、Sun NFS サーバのデーモンとして動作する。将来的には、この部分をモジュール化して他の NFS を解釈するモジュールに付け替えることによって、別の NFS プロトコルを使用できるようにする。例えば、ここに SMB や appletalk を理解するモジュールを使用すれば、Windows や Macintosh などでも、大きな改造をすることなく xxfs の機能を享受することができる。これは、xxfsd とファイル操作要求受付部の間のインタフェースを定義することで実現する。

6.2.2 マウント受付部

Sun NFS では、実際のファイル操作のためのプロトコル [1] と、Sun NFS を提供するファイル空間を拡張 (mount) するためのプロトコル [1] がそれぞれ定義されている。

カーネルから mount の要求があると、xxfsd 中のマウント受付部が Sun NFS の mount プロトコルに応じた返答を返し、その情報に基づいてカーネルは xxfs が提供するファイル空間を mount する。具体的には mount のリクエストがあると、xxfsd は xxfs ルートディレクトリのファイルハンドルを返す。

Sun NFS では、実際にはファイルを特定するために使う番号である NFS ファイルハンドル はファイルに対して固有の値を取る。このため、ファイル操作をするデーモンと、ファイル空間をマウントする際にマウントするトップディレクトリの NFS ファイルハンドル を提供するデーモンは別のものでも支障はなかった。しかし xxfs のシステムでは、ファイルシステムに特殊な機能を付与する場合もあるので、必ずしも NFS ファイルハンドル がファイルに対して固有の値を取るとは限らない。このため、ファイル操作部分とマウントを許可する部分で同じ情報を共有するためには、同一のプロセスで動作すると都合が良い。ここでは、以上を考えてファイル操作とマウント許可は同じプロセスで動作するようにした。

6.2.3 xxfs サーバ登録要求受付部

xxfs サーバ が起動する時に xxfsd に対して xxfs サーバリスト への登録を要求するが、それを受け付ける部分である。xxfs サーバ登録要求受付部は、xxfs サーバ からの登録要求を受け付ける

と、xxfs サーバリスト へエントリを追加する。また、この時に xxfs サーバ の番号付けも行なう。

xxfsd で扱う NFS ファイルハンドル もしくはその他の識別子には、ここで行なった番号付けに基づいた数字が使用される。xxfsd はこの番号を使用して xxfs サーバ の特定を行なう。NFS クライアントから xxfsd にファイル操作要求があると、その要求中に存在するファイル識別子を確認し、その一部に入力されている xxfs サーバナンバによって、目的の xxfs サーバ へファイル操作要求を転送する。

6.2.4 xxfs サーバへの要求発信部

xxfsd がファイル操作要求を受信した後に、あらためて xxfs サーバ へファイル操作要求を転送するためのインタフェースとなる部分である。

ここでは、ファイルが存在するディレクトリに応じて、xxfs サーバのリストより適切な xxfs サーバ を判断する。その判断を元に、実際に xxfs サーバ と通信を行なう。xxfsd と xxfs サーバとの通信路を常時を確保しているかどうかは xxfs サーバ の機能の性格による。

現在はこの部分は xxfsd の一部として固定しているが、将来的には xxfsd の発信モジュールの一つとして内部的に独立させる。これは、xxfsd が持つ機能を xxfs サーバ との通信に限定させないようにするためである。例えば通常の Sun NFS の nfsd と同様の働きをさせることで、xxfsd は単純に nfsd と置き換えて使用することができるようになる。なお、現在は RPC のプログラム番号の問題によって、xxfsd は Sun NFS の nfsd とは共存して同じホストに同居することはできない。

これは、RPC はポートマッパに登録されたプログラム番号によって、通信要求を行なったプロセスがどのプロセスとの通信を望むかを判断しているためである。主な Sun NFS の実装では、このプログラム番号はあらかじめ決められている。このため、Sun NFS プロトコルを利用するファイルシステムを製作する際には、Sun NFS が使用するとあらかじめ決められたプログラム番号を登録する必要がある。しかし、RPC では同一のプログラム番号[1]を持つプロセスを同時に登録することはできないので、xxfsd を登録した場合には、nfsd は登録できない。

ところで、例えば新しい機能を持つファイルシステムを xxfs サーバ としてでなく xxfsd の一部として動作させることができるようにすると、ネットワーク通信のオーバーヘッドが減少する分だけ、より高速に動作する。xxfsd の適切なモジュール化により、xxfs サーバへの要求発信部のかわりに新しい機能を持つファイルシステムの実装を導入可能にすれば、開発者やユーザにとってより便利になる。

6.2.5 メインモジュール

xxfsd のメインモジュールでは xxfs サーバ の管理と xxfs サーバ の特定を行なう。

xxfs サーバ に関する情報は xxfsd 内部にリストの形でデータを保持する。xxfs ルートディレクトリの構造はこのデータに依存し、xxfsd が xxfs サーバ のみに対応する場合は、当該 xxfsd に登録要求をしてきた xxfs サーバ の数だけディレクトリが存在することになる。

---

[1] 厳密には、tcp と udp で同じプログラム番号を同時に使用することはできる。

結果として、xxfs サーバ の登録要求に応じて xxfs のファイル空間のルートディレクトリは変化する事になる。xxfs サーバ から登録の要求があれば xxfs のファイル空間のルートディレクトリに xxfs サーバ の登録名を名前に持つディレクトリが出現する。

xxfsd のメインモジュールの機能として、将来的には以下の項目を実現する。

- モジュール化の促進

  特にファイル操作要求を受ける部分とファイル操作要求を転送する部分のモジュール化のインタフェースを厳密に定義し、モジュール化を進める。

- ファイルのキャッシュを備える

  ネットワークを通じてファイルの送受信を行なうのは時間的なコストが大きいため、これを避ける目的でキャッシュを設ける。これをすることで、新しいファイルシステムの開発者は独自にキャッシュを持たせることを考えなくても良い。

- ファイルハンドルとファイル名をマッチングさせるルーチンを備える

  現状では xxfsd は主にファイルハンドルによってファイルの特定を行なっている。しかし、新しいファイルシステムの開発者がファイルハンドルを解決するルーチンを作るのに労力が必要である。サービスライブラリとしてファイルハンドルを入力することでファイル名を返す、またはその逆の機能を持つルーチンを備える。これによって、ファイルシステムの開発に必要とする労力が軽減する。

6.2.6 xxfsd について

xxfs の主要部はユーザ空間上でデーモン (xxfsd) として動作し、NFS クライアントと NFS プロトコルを通じて通信する。xxfsd は仮想的なファイル空間として、現在提供している新しい機能を持つファイルシステムのリストを提供する。

NFS プロトコルに既存の Sun NFS や SMB などのネットワーク NFS プロトコルを利用することで、現行の NFS クライアントに大きな変更を加えずとも xxfs が想定する機能を実現することができる。

また、ネットワークを通じて他のホストで動作している xxfsd を利用することにより、xxfsd が動作していない、もしくは技術的に動作させるのが難しいホストで xxfs を利用することもできる。

xxfsd は他のプロセスとして動作している xxfs サーバ と通信する機構を持つ。これは、xxfs として動的に新しいファイルシステムを導入するためには、別のプロセスとの通信を行なう必要があるためである。

一つの xxfsd に対して xxfs サーバ は複数扱えるようあるべきである。このため、xxfsd 内部で現在使用可能な xxfs サーバ のリストを保持している。xxfsd はカーネルからの要求に応じて、xxfs サーバのリストを参照し、必要な xxfs サーバ との通信を確保する。

他の xxfs サーバ との通信には、新規に制定した xxfs プロトコル を用いる。ここに既存のプロトコルを用いなかったのは、カーネルと xxfsd との通信に既存のプロトコルを用いると、xxfs

としての柔軟性が失われてしまうという理由である。また、例えばファイルハンドルなどの単一のファイル指定方法に拘泥すると、そのプロトコルに合わせるための xxfs サーバ の開発者の手間が増える。これは xxfs プロトコル の方でファイルの指定方法が単一でないよう設計しておけば、新しい機能を持つ xxfs サーバ を開発する際に好きな指定方法を使用して開発することができるようになる。例えば、Sun NFS ではファイルの指定に NFS ファイルハンドル を用いるが、wwfs など FTP を利用するファイルシステムではファイルの指定にファイル名を直接用いる。

xxfsd 自体をモジュール化して、ライブラリとして新しいファイルシステムやカーネルと通信するプロトコルを付加させられるようにすることで、xxfsd の柔軟性を向上させることができる。例えば、xxfsd 自身に Sun NFS のデーモンとしての機能を持たせれば、そのまま xxfsd を nfsd と交換することで、今までの機能をそのままに xxfs の機能を利用できる。また、xxfsd のカーネルとの通信をするネットワークファイルシステムプロトコルを SMB などに変更することによって、特に変更無しに Windows98/NT などで xxfs の機能を使用することもできるようになる。

6.3 xxfs サーバの構造

xxfs サーバでは xxfs プロトコルスタック 以外の部分については特に制約は設けていない。
xxfs サーバは大きく下のように分類される

- xxfsd からの要求受付部
- xxfsd への登録要求部
- xxfs サーバファイル操作部

xxfs サーバ の構造を図 6.3に示す。

6.3.1 xxfsd からの要求受付部

xxfsd からのファイル操作要求受付部 はファイルに対する操作機能毎に関数を呼び出す。例えば、ファイルの読み出しを行なう要求をカーネルが行なった場合には最終的に xxfs サーバ の xxfs プロトコルスタック が xxfs サーバファイル操作部のファイル読み出し関数を呼び出す事になる。xxfs サーバファイル操作部の実装者はその関数に応じた機能を実装すればよい。

6.3.2 xxfsd への登録要求部

xxfs サーバは起動時に管理してもらう xxfsd に対して自身の登録要求を行なう。xxfsd への登録要求部は実際に xxfs サーバ が xxfsd に対して登録要求を発行する部分である。

起動時に要請を発行するのは当然だが、それ以降も xxfsd が再起動した場合やネットワークの不具合により通信不能になった場合も想定して、定期的に登録要請・確認を行なう。

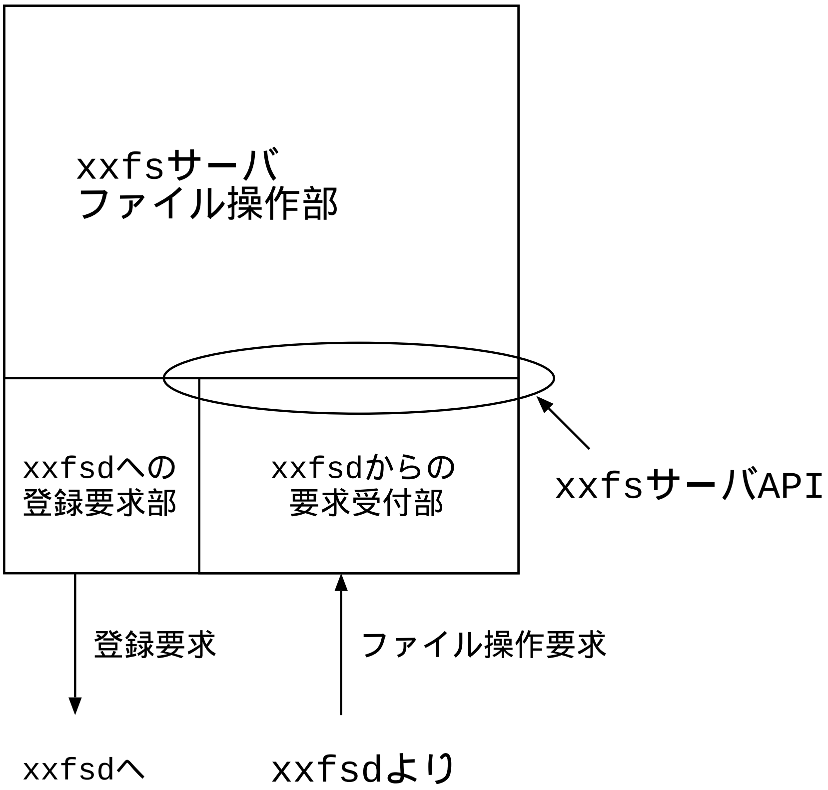

図 6.3: xxfs サーバの構造

6.3.3 xxfs サーバファイル操作部

ファイル操作部では、ファイル操作要求に応じて適切な操作をファイルに対して行なう。ユーザ権限でのファイル空間を拡張する機能を期待する場合は、この部分はファイル操作要求を単純に解釈し、その通りの操作を目的のファイルに対して行なう。

しかし、新しい機能を持つファイルシステムモジュールとして設計する場合には、ファイル操作要求に応じた別の機構を呼び出すことになる。新しいファイルシステムを設計する際には、読み出しや書きだしなどの動作に対応させてファイルシステムの機能抽象化をする必要がある。ファイル操作要求に応じて、その機能に抽象化された機能を実現するための機構を動作させる。

xxfs サーバライブラリ

以下に xxfs サーバメイン部へのインタフェースとして準備している関数の一覧を記述する。

```
attrstat * xxfsvfs_getattr( nfs_fh * NFSfh, struct svc_req * cl );
attrstat * xxfsvfs_setattr( sattrargs * SetAttrArgs, struct svc_req * cl );
diropres * xxfsvfs_lookup( diropargs * DirOpArgs, struct svc_req * cl );
readlinkres * xxfsvfs_readlink( nfs_fh * NFSfh, struct svc_req * cl );
readres * xxfsvfs_read( readargs * ReadArgs, struct svc_req * cl );
attrstat * xxfsvfs_write( writeargs * WriteArgs, struct svc_req * cl );
diropres * xxfsvfs_create( createargs * CreateArgs, struct svc_req * cl );
nfsstat * xxfsvfs_remove( diropargs * DirOpArgs, struct svc_req * cl );
nfsstat * xxfsvfs_rename( renameargs * RenameArgs, struct svc_req * cl );
nfsstat * xxfsvfs_link( linkargs * LinkArgs, struct svc_req * cl );
nfsstat * xxfsvfs_symlink( symlinkargs * SymlinkArgs, struct svc_req * cl );
diropres * xxfsvfs_mkdir( createargs * CreateArgs, struct svc_req * cl );
nfsstat * xxfsvfs_rmdir( diropargs * DirOpArgs, struct svc_req * cl );
readdirres * xxfsvfs_readdir( readdirargs * ReadDirArgs,struct svc_req * cl );
statfsres * xxfsvfs_statfs( nfs_fh * NFSfh, struct svc_req * cl );
```

現状では、返り値や引数に渡す構造は、Sun NFS Ver.2 に準じた形になっている。

6.3.4 xxfs サーバについて

xxfs サーバ は、xxfsd からの要求に応じて実際にファイルの操作・参照を行なうデーモンである。ユーザが自由にファイル空間を xxfs を利用するホストに提供できるようにするため、xxfs サーバ はユーザ権限で動作可能である。

xxfs サーバ は要求に対して単純にファイルの操作・参照を行なうだけでなく、ファイル操作に別の機能を付加することも考えられる。例えば、リビジョンコントロールシステムをファイルシステムに統合してしまうことで、ci/co などの特別な手続きを必要とせずにファイルのバージョン

管理をすることができる。また、SMTP や HTTP などのネットワークサービスをファイルシステムに抽象化・導入することで、様々なネットワークサービスをファイルシステムを通じて使用することもできるようになる。xxfs サーバ では通信のためのプロトコルスタックも用意してあるので、新ファイルシステムを開発する際には実際のファイル操作部のみ作成すればよいようになっている。

なお、xxfsd は一つの xxfsd で複数の xxfs サーバ を管理することができるが、xxfs サーバ は一つの xxfsd とのみ通信することができる。

6.4 動作概要

ここでは、実際に xxfs を動作させる際に xxfsd や xxfs サーバ などがどのように動作するのかを簡単に説明する。

xxfs の動作概要を簡単に図 6.4 に示す。

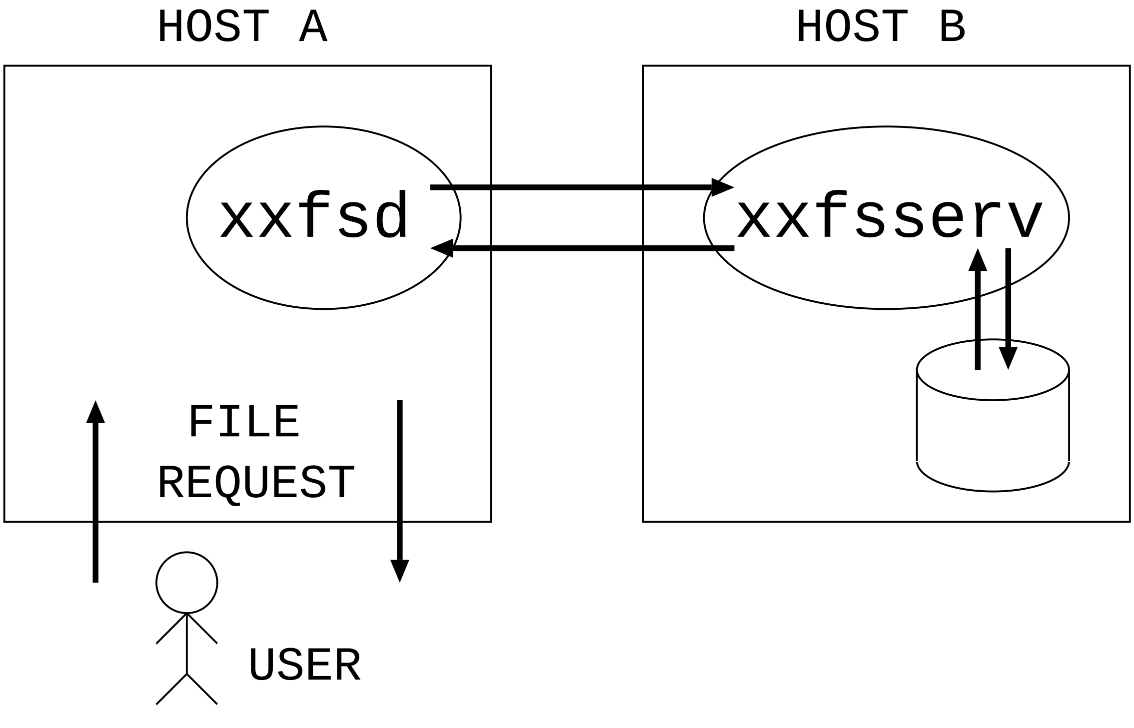

図 6.4: xxfs の動作概要

6.4.1 マウント

xxfsd が提供するファイル空間を利用するためには、利用するホスト側で mount システムコールを発行する必要がある。このため、xxfs は xxfs をマウントするためのマウントポイントが必要である。また xxfsd によってサービスされているファイル空間のルートディレクトリを用意したマウントポイントにマウントする必要がある。

具体的には、例えば /mnt/xxfs というディレクトリを用意し、下に示されるようなコマンドを実行する。

```
# mount -t nfs localhost:/ /mnt/xxfs
```

なお、3 つ目の引数に存在するマウントポイント、上記例では localhost: の後に付く / は現在はどのポイントを指定しても動作に変化はない。これは将来的にはなにがしかの意味を持たせる事も考えている。

6.4.2 デーモン部

xxfsd は起動後、xxfs サーバ からのファイル空間への登録要請とカーネルからのマウント要求・カーネルからのファイル操作要求を待つ。

カーネルからのマウント要求があった場合には、xxfsd が管理するファイル空間のルートディレクトリ (xxfs ルートディレクトリ) へのファイルハンドルを返す。

xxfs ルートディレクトリには、xxfs サーバの登録名がディレクトリ名になっているディレクトリが羅列される。つまり、xxfs ルートディレクトリには現在 xxfsd が管理している xxfs サーバの個数だけディレクトリが存在する。なお、この xxfs ルートディレクトリに対しては mount しているホスト側からの書き込み・削除等はできない。

xxfs サーバ からファイル空間への登録要請があった場合には、ユーザ認証フェーズなどを経て、xxfsd 内部に持つ、ファイル空間ポイントを保持するデータリストに xxfs サーバ のホストやポートなどの情報を登録する。実際の xxfs サーバ との通信はこのリストに登録されている情報を元に行なう。カーネルなどからファイル操作の要求があった場合に、xxfs サーバのリストを参照して、要求ファイルが存在するディレクトリに応じて適切な xxfs サーバ と通信する。

xxfs サーバ の対応付けには、現状では Sun NFS ファイルハンドルの上位 4bytes を利用している。xxfs サーバが登録要求をしてきた時点で、4bytes の名前を与える。xxfsd はカーネルから要求があったファイルについてファイルハンドルを解釈し、その上位 4bytes に応じた xxfs サーバと接続する。

6.4.3 サーバ部

xxfs サーバ は動作を開始すると、引数等から判断して、目的のホストで動作している xxfsd に当該 xxfs サーバ が管理するファイル空間の登録要求を行なう。

xxfs サーバ は xxfsd に対して xxfsd が管理するファイル空間のどこにマウントするかという情報や、xxfsd がカーネルに対して提供する際のユーザの権限についての情報などを渡す。

一度登録要求を送ってしまった後は、xxfs サーバ は xxfsd からファイル操作要求があるまでは待機状態になる。そして、xxfsd から要求があると、その操作に応じた内容の動作を行ない、結果を xxfsd に応答する。xxfsd はその結果を適切に処理し、カーネルに送付する。

このサーバ部には単純に要求に応じてファイルを操作するだけでなく、ファイルの操作に対応付けして、ネットワークサービス等をファイルシステムに抽象化する事ができる。例えば、ある xxfs サーバ に ftp スタック を実装しておき、xxfsd のリクエストに応じて、ftp スタック を動作

させることで、疑似的に ftp 先をマウントしてユーザ側からはあたかもローカルファイルを使用しているかのような操作をすることもできるようになる。また、例えば SMTP を適切にファイルシステム操作に抽象化する事で、ファイルを書き出す事で適切な相手に対しメールを送信する事もできるようになることも考えられる。

6.5 実装

現在、xxfsd は複数の xxfs サーバ を制御できる機構を備えている。また、xxfs プロトコルに応じて単純にファイルの操作を行なう xxfs サーバ を実装した。この xxfs サーバ をあるホストで動作させ、xxfsd を通じて動的に mount/umount ができるようになっている。この段階では、カーネルとの通信部は単一のプロトコルのみに対応し、xxfsd の機構は xxfs サーバ と通信するために特化されている。

xxfsd とカーネル部の通信には Sun NFS version 2 を利用し、Linux 2.0.3x(slackware3.5) 上で xxfsd が動作し、xxfs サーバは別の管理形態をしている solaris2.6 上で一般ユーザ権限で動作するよう実装した。xxfs プロトコルは、現在は Sun NFS version 2 に準拠した作りになっており、ファイルの指定には Sun NFS ファイルハンドルを使用している。

Linux から Sun NFS を使用して xxfsd をマウントし、xxfs サーバから登録要求を送信すると、Linux 側から solaris 上に存在するファイル空間を表示したり、ファイル操作したりすることができるようになっている。

表 6.1: 実装環境

	xxfsd	xxfsserv	xxfsserv
計算機	IBM-PC/AT 互換機	IBM-PC/AT 互換機	SPARCstation 20
オペレーティングシステム	Linux 2.0.36	Linux 2.0.36	SunOS Release 5.6 Version Generic [UNIX(R) System V Release 4.0]
言語	ANSI C	ANSI C	ANSI C
処理系	egcs-2.91.57 19980901 gcc-2.7.2.3	egcs-2.91.57 19980901	gcc-2.7.2.3

6.5.1 実際の使用例

ネットワーク上に複数のアカウントを所持しており、そのうち一つを使用している際に、別のアカウントを使用している時に実装したファイルを参照・操作する場合、 xxfs を用いると楽に作業を続けることができる。

以下に実際の使用法を具体的に記す。

まず、自分が使用しているホストで xxfsd を動作しており、管理者によって xxfsd によってサービスされているファイル空間をマウントされている。この時にはまだ xxfsd にはファイル空間は登録されていないため、xxfsd が管理しているディレクトリの中を見ても何も登録されていない。

次に xxfs サーバ を、操作するファイルの存在するホストで動作させる。この際、一度操作するファイルの存在するホストで xxfs サーバを実行するためのコマンドを実行する必要がある。

このコマンド行の例を下に挙げる。

```
% xxfsserv gulf.sfc.keio.ac.jp gulf
```

この例では、gulf.sfc.keio.ac.jp で動作している xxfsd に対して gulf というエントリー名で xxfs サーバ が登録される。

xxfs サーバ が立ち上がると、xxfs サーバは自分の使用しているホストで動作している xxfsd に対して登録要求する。この際 xxfsd は、パスワード認証などにより、登録リストに登録して良いか判断する。

xxfsd はこの時点で xxfsd が管理するルートディレクトリに新たなディレクトリを追加する。なお、この xxfsd が管理するルートディレクトリ部分についてはユーザがディレクトリを作ることやファイルを作ることはできない。

自分が使用しているホストで xxfsd によって管理されているディレクトリにカレントディレクトリを移し、ディレクトリの中を見ると、xxfsd は xxfs サーバ に対してディレクトリ閲覧の要求を発行する。xxfs サーバ は要求に応じて当該ディレクトリの内容を応答し、その結果がローカルホストのコマンドライン等に返される。

このような状態でファイルを操作すれば、通常の Sun NFS を利用して共有しているファイルシステムであるかのように扱うことができる。

6.5.2 複数クライアントでの使用

xxfs サーバ を xxfsd に登録する際にファイル空間のエントリー名を指定し、それによって xxfs が提供するファイル空間の ユーザ ID とグループ ID、パーミッション を決定する。このユーザ ID とグループ ID は xxfsd の設定ファイルによって設定される。パーミッションは xxfs サーバによって提供される側のパーミッションと設定ファイルで設定されたパーミッションの AND を取って、それを実際のパーミッションとして使用する。

xxfsd は複数の xxfs サーバを管理することができる。このため、複数ユーザがそれぞれ xxfs サーバ を xxfsd に登録して同時に使用することが可能である。この際、ユーザ権限の解決は ユーザ ID と グループ ID、パーミッションを設定することによって行なう。xxfsd が動作しているホストの設定ファイルをそれぞれの xxfs サーバに対して適切に記述することで、複数のユーザが使用可能である。

xxfs の設定ファイルの記述法は、「 A.2 xxfsd.conf の説明 」を参照。

6.6 開発者の視点

xxfs では新たなファイルシステムを実装しやすくする事を目的の一つとしている。ここでは新たに付加価値を付けたファイルシステムを実装しようとする場合には、xxfs のどこの部分に新しい機構のファイルシステムを実装するべきかを考える。

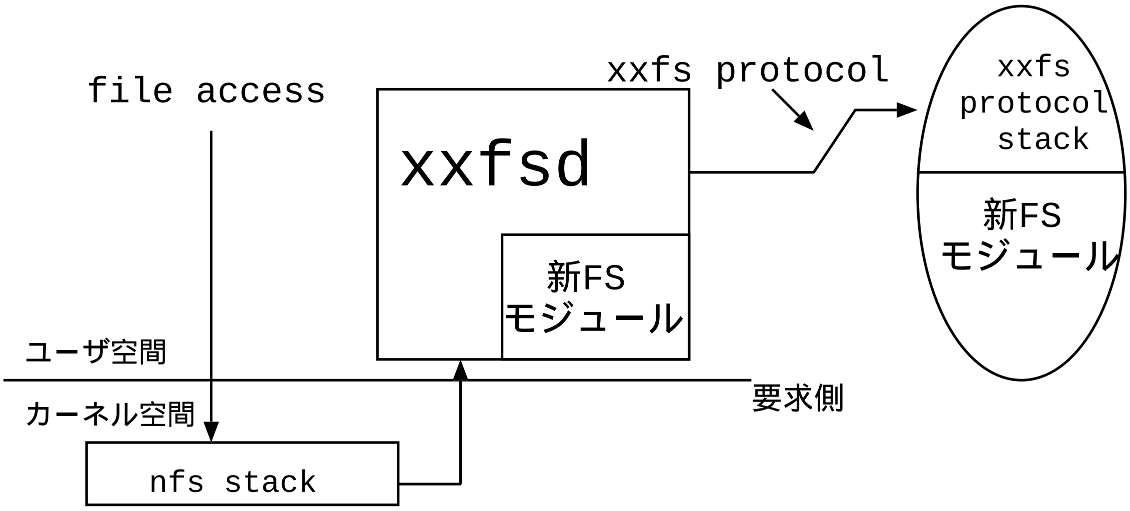

図 6.5: 新 FS を組み込む位置

xxfs サーバとして実装

まず、一番最初に考えられるのは新しい機構のファイルシステムを xxfs サーバ として動作させる事である。xxfs プロトコルスタック を実装し、新たなファイルシステムの機構をファイル操作に抽象化して結合する事で、新たな機構を持つファイルシステムとして動作をできるようになる。ただし、この手法では一度 xxfsd とカーネルが通信した後、更に xxfsd と xxfs サーバ とが通信する必要があるため、高速な動作は期待できない。開発初期においては容易に再起動可能な xxfs サーバ として実装する方が容易に開発できる。

xxfsd に組み込む

このため、二番目に考えられるのは、xxfsd にモジュールとして組み込むことである。これは、xxfs サーバ を実装する場合と違い、デバッグの手間が増えるが、通信のオーバーヘッドはなくなる。

これらの実装位置について、それぞれ図 6.5に示す。xxfs プロトコル を通じて xxfsd と通信しているものが、xxfs サーバ として動作させた場合であり、xxfsd 内部に存在するのが、xxfsd にモジュールとして組み込む場合である。

Chapter 7

xxfs の評価

7.1　性能評価

幾つかの類似研究とファイルの転送速度について比較する。

ファイルの転送についてその構造上、大きいサイズのファイルを転送する場合と小さいサイズのファイルを転送する場合では要求される性能は大きく異なる。このため、今回の性能評価では大きいサイズのファイルを転送する場合と、小さいサイズのファイルを大量に転送する場合の二つの実験を行なった。

7.1.1　実験環境

この実験では、ホスト jetta から ホスト beetle へのファイルの転送を行なう。それぞれのホストの環境は表 7.1の通りである。

表 7.1: ホストの環境

	jetta	beetle
CPU	Genuine Intel Celeron 333MHz(375MHz 駆動)	Genuine Intel mobile Pentium MMX 233MHz
メモリ	SDRAM 128MBytes	EDO 64MBytes
Ethernet Card	ISA NE2000 互換カード 10base-T	Allied Telesis PCMCIA 10base-T
OS	Linux 2.0.36	Linux 2.0.36

これらを図 7.1のように接続し、実験環境とする。なお、このネットワークには他のホストは接続されておらず、jetta - beetle 間以外の通信は行なわれていない。

いずれの実験でも、シェルとコマンドの実行は beetle 側で行なう。なお、転送のコマンドには cp を、時間の計測には time を利用した。

また、それぞれの実験の間には転送したファイルの削除を行なっている。これはファイルシステムがリモートホストとローカルホストのファイルの同一性を確認する場合考えている。転送前のファイルが残っている場合、ファイルシステムはリモートホストとローカルホストのファイル内容が同一であると認識し、実際にはファイルが転送されないため、誤った結果が出てしまう。

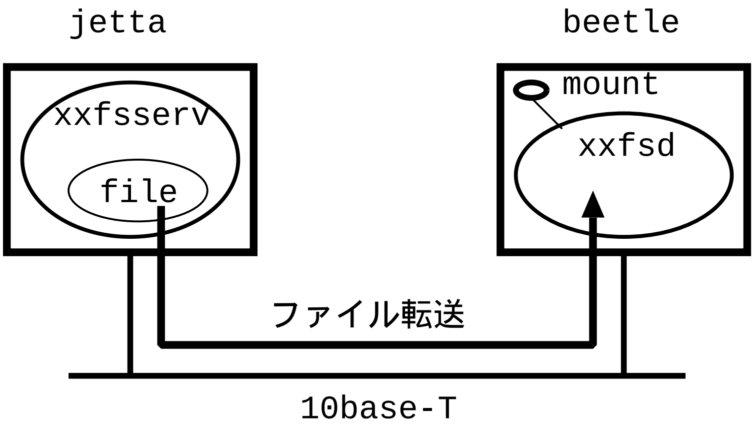

図 7.1: 実験環境

表 7.2: 大サイズファイルの転送 (分:秒)

使用ファイルシステム	1 回目	2 回目	3 回目	4 回目	5 回目	平均
xxfs	3:37.27	4:49.84	4:25.41	3:31.53	3:24.56	3:57.72
Sun NFS	2:53.32	2:50.04	2:51.01	2:50.71	2:51.50	2:51.32
参考:ローカルホスト上でのコピー	0:24.56	0:28.23	0:28.16	0:28.15	0:28.86	0:27.59

7.1.2 測定

大サイズファイルの転送

ここでは、50Mbytes のファイルを jetta から beetle に転送する。それぞれ 5 回転送を繰り返し、それぞれの時間と平均時間を取った。

結果を表 7.2に示す。

表 7.2から見られるように、ファイル操作のオーバーヘッドが大きくなる分 xxfs は Sun NFS より遅い動作となった。

大量のファイルの転送

次に小さいサイズのファイルを大量に転送する。具体的には、5Kbytes のファイルを 100 個 まとめて転送することで、計測した。

結果を表 7.3に示す。

表 7.3: 大量のファイルの転送 (分:秒)

使用ファイルシステム	1 回目	2 回目	3 回目	4 回目	5 回目	平均
xxfs	0:53.13	0:06.97	0:06.59	0:07.31	0:04.08	0:15.62
Sun NFS	0:34.13	0:04.62	0:04.50	0:05.45	0:03.29	0:10.40
参考:ローカルホスト上でのコピー	0:02.26	0:01.71	0:01.70	0:01.71	0:01.90	0:01.86

特に xxfs と Sun NFS では、最初の 1 回目に時間がかかるが、2 回目以降は短時間で終了する

ことがわかる。これは、取得ホスト側 (beetle) にファイルがキャッシュの形で残っており、実際には、2 回目以降はネットワーク上をファイルの転送は行なわれていないためである。

7.1.3 評価

ここでは、Sun NFS という今日有力なファイルシステムに対して、ファイル転送の能力について比較を行なった。結果として Sun NFS と比較して速度が遅いことは否めないが、充分に実用に耐える範囲内であるといえる。更に、Sun NFS にはないユーザ権限でのファイルシステムの導入やファイル空間の拡張が行なえることを考えると、xxfs は充分に魅力のあるシステムである。

7.2 定性的評価

ここでは、xxfs を利用することで、どの程度新しい機能を持ったファイルシステムの開発者の負担が減るか調べる。対象として、wwfs を例にとる。

wwfs を全く元になるものがない状態から書き始める場合を考える。これは、現在入手できる形の wwfs がそれにあたる。これを展開し、wwfs の主要部を残して、説明書や wwfs を利用するためのユーティリティコマンドなどを除去した部分のプログラムソースの合計量を計測した。また、これに対する値として、xxfs を使用して実装すると仮定し、xxfs が提供している機能の部分を wwfs のソースから除去した場合のプログラムソースの合計量を合わせて計測した。wwfs のソースを調べて、ソースファイル毎に xxfs を利用する場合には必要であるか必要でないかを判断した。なお、ここでは xxfs によるキャッシングなどの、現在は提供できていないが将来的に提供する機能も、xxfs を利用する場合には必要でないものとした。

これらの結果を合わせて表 7.4に示す。

表 7.4: xxfs を未使用の場合の wwfs ソース量

	行数	バイト数
xxfs 未使用時	23609 行	658287 bytes
xxfs 使用時	9134 行	279236 bytes

この例を見ても分かる通り、新しい機能を持つファイルシステムの開発に xxfs を利用することでかなりの量のプログラムソース量が減る。この例では特に、NFS の機種依存関係の定義ファイルが多量に存在したために差が良く目立ったが、これ以外の新しい機能を持つファイルシステムの開発でも、費やす労力は大幅に減少する。

つまり、xxfs は新しい機能を持つファイルシステムの開発に大いに役に立つシステムであると言える。

7.3 問題点

設計上および構造上幾つかの問題点が考えられる。

ここでは、これら問題点に対して解決の方向について説明する。

7.3.1 再起動性

現在の実装では、xxfs ではカーネルと xxfsd の間の通信に Sun NFS プロトコルを使用する。Sun NFS プロトコル の特徴の一つに、nfsd はクライアントの状態を保存しない事があげられる。このことで、nfsd が中途で動作しなくなったとしても、再度 nfsd を立ち上げ直すことによりクライアント側は問題なく使用を続けることができる。

しかし、xxfsd は xxfs サーバ の状態を保存する。このため、一度 xxfsd を再起動した場合にはこれらの情報は消える。これにより、Sun NFS プロトコルの特徴である耐障害性に支障をきたす。これは xxfs サーバ が定期的に xxfsd に登録の確認を行なうことで復帰が可能であるが、復旧までに若干時間が必要になるという欠点を持つ。

7.3.2 NAT または IP マスカレード での使用

xxfsd に対して、xxfs サーバ が自身の登録を行なうために、通信路を確保する。しかし、xxfsd が NAT(Network Address Translation) もしくは IP マスカレード によって管理されるプライベートネットワークに存在する場合がある。この場合、インターネットに存在する xxfs サーバ からプライベートネットワーク内に対して通信路を確保することが出来ないため、xxfsd に登録ができない。

これに対応するためには、逆に xxfsd が動作しているホスト側から通信路を確保する機構を実装する必要がある。しかし、この機能を実装するためには、動作中の xxfsd をユーザ権限で直接操作する必要が出てくるため、実装には細心の注意が必要となる。

7.3.3 権限について

自由にユーザがファイル空間を拡張できるようになると、当然のことながら使用権限を考慮しなくてはならない。

xxfs サーバ を使用する事を考えると、xxfs サーバ が動作するホストのユーザ管理形態と xxfsd を使用してマウントしているホストのユーザ管理形態は異なる事が多い。この場合、例えば xxfs サーバ からリクエストによってマウントされたファイル空間は誰の権限で操作できるようにしておくかという問題が生ずる。

結局 xxfs サーバ を起動する時点で xxfsd に対してどのユーザ権限でファイル空間に追加するかを指定する必要が出てくる。これは、設定ファイルで行なうか、xxfs サーバ を立ち上げる際にパスワードなどによる認証をする等の機能を xxfs プロトコルに実装する必要がある。

7.3.4 セキュリティについて

基本的には xxfs サーバ は一般ユーザ権限で動作することで xxfs サーバ が動作する側では xxfs サーバ を動作させたユーザ権限以上の操作をすることはできない。

しかし、ホストに存在するの多くのファイルの読み込みは一般ユーザ権限でできるため、例えば passwd ファイルを含むファイル空間を export された場合には、xxfsd をマウントしているユーザから xxfs サーバ が動作するホストの passwd ファイルが読める。このため、xxfs サーバ

が動作するホストにアカウントを持たず、xxfsd をマウントしているホストにはアカウントを持つユーザが passwd ファイルに記述されたパスワードエントリを crack するという可能性がある。

また、例えば、xxfs サーバ を実行するユーザがルートディレクトリを export したとすると、xxfsd を用いてそのファイル空間をマウントしているサーバからは xxfs サーバ を実行しているホストの設定ファイルの多くを読むことができる。

xxfs サーバ自体がそのようなディレクトリをマウントできないようにするという手法も考えられる。しかし、xxfs 自身がファイルシステムを容易に開発できることを目的としているため、ユーザが xxfs サーバ を開発することも考えなくてはならない。

xxfs サーバ が許可したユーザ以外のユーザは、xxfs サーバ経由でディレクトリやファイルを読めないようにするような機構を xxfsd もしくは xxfs プロトコル に内包させる事が要求される。

Chapter 8

結論とまとめ

8.1 結論

xxfs はファイルシステムを容易に開発・導入できるようにするための仕組みである。この仕組みを実現するために、本研究ではファイルシステムモジュールとなる xxfs サーバ を管理するためのプロセスである xxfsd を作り、xxfsd を通じてファイル操作をする仕組みを作成した。

xxfs サーバからの要求により、xxfsd は動的に xxfs サーバ が管理するファイルへの経路をディレクトリとして拡張する。これにより、mount システムコールによらずにファイルシステムを導入することができるようになる。この手法を使用することで、ユーザ権限しか持たないものでも、ファイルシステムを導入することができるようになった。同時に、ファイル空間もユーザ権限で拡張できた。

また、ローカルホストとリモートホスト間の管理状況などの違いを吸収するための仕組みを導入し、それをファイルシステム開発者に提供することで、新しいファイルシステムの開発にかかる労力を減少させることに成功した。

8.2 今後の予定

今後、xxfsd をモジュール化し、カーネルとの通信部とファイルへの操作要求部をそれぞれ容易に新しいプロトコルに変更できるようにする。これによりカーネルの通信部を、例えば SMB や appletalk を使用できるライブラリを導入する事によって、これらのプロトコルを用いた通信をすることができるようになる。また、操作要求部に xxfs プロトコル を用いた通信をしないモジュールを使用することによって、ローカルファイルシステムに対するアクセスを高速化させることができる。xxfs プロトコルスタックはこの時点で xxfsd の数あるモジュールの中の一つに過ぎない扱いとなる。

上記のシステムでは xxfsd にモジュールを追加するためには xxfsd の再構築を行なう必要があるが、これを動的に結合するようにして、新たなファイルシステムを付加したりカーネルとの通信プロトコルを追加したりできるようにする事で、システムに柔軟性を与える事ができる。

また、xxfsd 内部にキャッシュの機構やファイル特定方法の変換機構を備える。この段階では主に高速化や高性能化を図る。

現況では構造的なセキュリティについてもまだ脆弱だが、ネットワーク上におけるスニファ対策などについては非常に脆弱であると言わざるを得ない。これをもっと頑強なセキュリティ構造を実装する必要がある。例えば、DES や RSA[6] などの使用できるライブラリを用いて xxfs プロトコルを隠蔽する方法や、kerberos[7, 8] などを用いてセッション毎に通信相手を保証する事などが考えられる。このようなセキュリティ面については構造面も含めてもっと改良してゆく余地がある。

今後はシステムの柔軟性を権限・セキュリティに破綻をきたさない限り最大限に引き出す事を考えて開発する。

また、現在私の所属する研究室では幾つかの mobile 環境を意識したネットワークサービスを考案・実装している。本研究はこれらネットワークサービスがファイルシステムを利用するものである場合に有用である。この中で、不安定なネットワークにおけるファイルの同一性を保証する研究がなされているが、これをこのファイルシステムと組み合わせる事でより使用しやすい環境を整える事ができる。

このシステムを利用して多くの有用なシステムが開発される事になるよう、公開を予定している。

謝辞

この研究を進める上で、御指導頂きました慶應義塾大学環境情報学部の徳田英幸教授と、同じく慶應義塾大学環境情報学部の村井純教授に感謝します。また、この研究の方向性を決めて頂き、その後も論文を書く上で、色々と助言を下さった慶應義塾大学環境情報学部助手の植原啓介氏, 慶應義塾大学政策・メディア研究科後期博士課程の渡辺恭人氏, 冨永明宏氏に感謝します。忙しい時間をやりくりして論文の構成を見て頂いた、慶應義塾大学政策・メディア研究科の土井裕介, 小浦大将, 東 秀明 諸氏、また論文を書く環境を壊さないよう気遣ってくれた学生コンサルタントの面々と徳田村井楠本中村共同研究室のみなさんに感謝の意を表します。

Bibliography

[1] SUN microsystems, Inc: NFS:Network File System Protocol Specification, RFC1094 (1989)

[2] Youki Kadobayashi: WWFTP: A filesystem-oriented file transfer protocol, WWFS Research Group (1993)

[3] 門林雄基, 山口 英, 宮原秀夫: Internet における資源アクセス装置の提案, 情報処理学会論文誌, Vol.37,No.5 (1996)

[4] 門林雄基, 山口 英, 宮原秀夫: インターネットにおける可用性改善のための代替サーバ利用技術, 情報処理学会論文誌,Vol.38,No.2 (1997)

[5] 楯岡孝道, 植原啓介, 砂原秀樹, 寺岡文男: PFS: 通信環境に動的に対応するファイルシステム, インターネットコンファレンス'96 論文集, 日本ソフトウェア科学会インターネットテクノロジ研究会 / 日本 UNIX ユーザ会 / WIDE プロジェクト (1996),pp.81-88

[6] RSA Laboratories: RSA Laboratories. FAQ 4.0, RSA Laboratories(1995)

[7] J.Kohl,C.Newman: The Kerberos Network Authentication Service (V5), RFC1510(1993)

[8] 稲村 雄: Kerberos, OPEN DESIGN No.14, CQ 出版社 (1996)

[9] 宅間 啓: パーソナル分散システム, 慶應義塾大学環境情報学部卒業論文 (1997)

[10] David S.H.Rosenthal: Evolving the Vnode Interface, USENIX Summer Conference, Anaheim, California (1990)

[11] John Bloomer, 森島 晃年監訳: RPC プログラミング, アスキー出版社 (1995)

[12] Hal Stern, 倉骨 彰訳, 砂原 秀樹監訳: NFS&NIS, アスキー出版社 (1992)

Appendix A

manpages

A.1 xxfsd の説明

`xxfsd(1)`

名称

`xxfsd - X X File System` デーモン

形式

`xxfsd [ -f config-file ]`

解説

`xxfsd` はクライアントのファイル操作要求を処理するデーモンです。起動時に `/etc/xxfsd.conf` を読み取り、`xxfsserv` によって登録されるファイル空間の `UID/GID` を決定します。`/etc/xxfsd.conf` に記述されていないファイル空間を登録する場合は、`UID/GID` はそれぞれ `nobody/nogroup` に設定されます。

オプション

`-f` 起動時に読み取るコンフィグファイルを`/etc/xxfsd.conf`から指定されたファイルに変更します。

関連ファイル

`/etc/xxfsd.conf`

関連事項

`xxfsd.conf(5), xxfsserv(1)`

A.2 xxfsd.conf の説明

`xxfsd(1)`

名称

`xxfsd.conf - X X File System Daemon Configuration File`

形式

`xxfsd.conf`

解説

`xxfsd` は `xxfsd.conf` ファイルによって、`xxfsserv` から登録されるファイル空間のユーザ権限を決定します。`xxfsd.conf` には、これを決定するための、相手ホスト名、ファイル空間エントリー名、UID、GID、パーミッションが記述されます。

ファイル空間エントリー名は `xxfs` によって提供されるファイル空間を識別するための名前です。`xxfsserv` を `xxfsd` に登録する際に指定します。`xxfsserv` によって新しいファイル空間が登録された場合、`xxfsd` によって提供されるファイル空間のトップディレクトリにエントリー名を名前に持つディレクトリが生成されます。

`xxfsd.conf` ファイルの形式について説明します。`#` で始まる行は、それ以下はコメントとみなされ、無視されます。
実際に必要となる要素は、ファイル空間エントリー名毎に整理され、それぞれに情報が書き込まれます。一通りの要素はファイル空間エントリー名に続く　の中に記述され、　の内部に記述されたものはその他のファイル空間には影響しません。
また、　の外に書かれた要素は、該当するファイル空間エントリー名が `xxfsd.conf` に存在しなかった場合のデフォルトの要素として判断されます。

`xxfsd.conf` に記述される要素は、ファイル空間エントリー名の他に以下のものが存在します。

`hostname`

- 相手ホスト名 または IP アドレス

`xxfsd` は相手のホスト名とファイル空間エントリー名の対によって、指定した要素が該当するかどうかを判断します。相手ホスト名が記述されていない場合は、ファイル空間エントリー名のみが一致すれば指定した要素を使用します。

ホスト名が記述されている場合に、ファイル空間エントリー名のみ一致した登録要求があった場合には、登録が拒否されます。

`UID` - ユーザ ID

自分のホスト側で、このファイル空間の所有権がどのユーザのものであるか設定します。省略された場合はデフォルトとして設定されたユーザ ID を、それも省略された場合は、`nobody` を使用します。

`GID` - グループ ID

自分のホスト側で、このファイル空間がどのグループ ID に属するかを設定します。省略された場合はデフォルトとして設定されたユーザ ID を、それも省略された場合は、`nogroup` を使用します。

`permission`

- パーミッション

自分のホスト側で、このファイル空間の操作権限をどの様に設定するかを設定します。省略された場合はデフォルトとして設定されたパーミッションを、それも省略された場合は、ファイル空間提供側のパーミッションを使用します。

なお、実効パーミッションにはファイル空間提供側のパーミッションとここで設定されたパーミッションを AND 演算したものが用いられます。

例

以下に、`xxfsd.conf` の簡単な例を挙げます。

```
# sample xxfsd.conf file
UID     nobody
```

```
GID     nogroup
permission      0775

sample
        hostname        yourhost.yourdomain.ac.jp
        UID     duck
        GID     users
        permission      0775
```

上記の例では、sample 以外のファイル空間エントリー名を指定された場合は、UID に nobody を、GID に nogroup を使用します。
また、yourhost.yourdomain.ac.jp から sample というファイル空間エントリー名で指定された場合のみ、UID に duck を、GID に users を使用します。

関連事項

xxfsd(1), xxfsserv(1)

A.3 xxfsserv の説明

xxfsserv(1)

名称

xxfsserv - X X File System サーバ

形式

```
xxfsserv [ -d directory ] hostname entryname
```

解説

xxfsserv はファイル空間を提供するホストで動作するファイルサーバです。起動時に config file を読み取り、提供するファイル空間と entryname の対を決定します。

オプション

`-d`	提供するファイル空間をディレクトリで指定します。 相対ディレクトリを入れても構いません。 省略した場合はカレントディレクトリが **`xxfs`** で提供されるファイル空間に追加されます。

関連事項

`xxfsd(1)`