第 13 部

広域分散ファイルシステム

第 1 章

はじめに

ネットワーク技術、特にインターネット技術の普及に伴い、世界規模での分散環境を構築する基盤が確立されてきている。このような、大規模広域分散環境において情報を共有するためのメカニズムの一つとして、これまで NFS[110] や Andrew File System (AFS)[111]、Coda[112] などの分散ファイルシステムが提案されてきた。これらは、ネットワーク上の他の計算機に配置されたファイルシステムを自分が利用する計算機のファイルシステムとして取り込むことにより、ネットワーク上のファイルを共有する機能を提供している。このような機能を利用することで、ファイルを共有し効果的な情報の配置が可能となっている。

しかし、分散ファイルシステムが利用できる範囲が広がるにつれて以下に示すような問題も生じるようになってきた。

- ファイルシステムの名前空間が非常に大きくなり、目的のファイルを探すことが困難となる。
- Sun ワークステーションから NEWS ワークステーション用の実行モジュールが見えるなど、利用者にとって不必要なファイルまでが見える。
- ファイル内での表現形式 (フォーマットや文字コード) を、供給する側と利用する側で整合性を取る必然性がある。
- セキュリティ上、存在さえも知られたくないファイルの存在が明かになる。

こうした問題を解決するためには、条件にしたがって共有されているファイルシステム中からファイルを見え無くするメカニズムが必要となってくる。そこで我々はファイルの可視性制御機構を提案し [113]、さまざまな応用についての検討を行ってきた。ここでは、提案する可視性制御機構について解説し、その実装例と応用について述べ、今後の課題をまとめる。

第 2 章

ファイルの可視性制御

複数のファイル実体とファイル名の関係を動的に制御し、条件にしたがって同一のファイル名で異なるファイルの内容にアクセスできる機能を**可視性制御機構**と呼ぶことにする。ここで、条件に対応するファイル実体が存在しないとき、ファイルは存在しないようにシステムは振る舞うことになる。

2.1 これまでの研究

複数のファイル実体とファイル名の制御を行う機構としてこれまで、以下のような機構が提案されている。

2.1.1 条件付きシンボリックリンク

カーネル変数や環境変数によって、シンボリックリンクが指示するファイル実体を切り替えるメカニズムである [114, 115]。NEWS-OS の実装の場合、シンボリックリンクの特別な形式で表され、以下のような形式を用いる。

expr?name1:name2

expr の部分には、 $env==val という条件を記述し、環境変数 env の値が val と等しい場合は name1 が、それ以外の場合は name2 が参照される。

例えば、機種名を設定する環境変数を利用する事で、先に述べた機種依存ファイルの選択を自動的に行う事も可能となる。

実際にコンディショナルシンボリックリンクを用いた例を以下に示す。

```
@bigbird% ln -s '$HOST'=="bigbird"?bigbird:hood hostname
@bigbird% ls -la
 -rw-r--r--  1 koni   13 Feb 14 03:00 bigbird
 -rw-r--r--  1 koni   10 Feb 14 03:00 hood
 lrwxrwxrwx  1 koni   27 Feb 14 03:02 hostname -> $HOST==bigbird?bigbird:hood
@bigbird% echo $HOST
```

```
bigbird
@bigbird% cat hostname
I'm bigbird.

@hood:% echo $HOST
hood
@hood:% cat hostname
I'm hood.
```

条件付きシンボリックリンクは、実装が容易であり利用も簡単であるが、条件の追加や変更などの作業が面倒であると共に、実体となるファイルとの関係が単方向でしか存在せずファイルの関係が不明確になりやすく、ループが発生する可能性 (図 2.1) があるなどシンボリックリンク特有の問題を引きずっている。

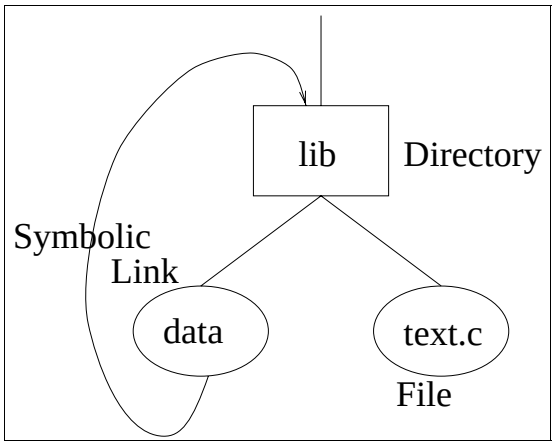

図 2.1: シンボリックリンクのループの例

2.1.2　3D ファイルシステム

ソフトウェア開発中に存在する複数のバージョンを管理するために開発されたファイルシステムである [116]。開発中のバージョンは開発者だけが管理し、配布用のバージョンは利用者に公開するなどの制御が可能である。

Version File

Version File はディレクトリであり、その内部にバージョン番号をファイル名とした複数のファイルを持つ。 Version File 自身の名前は、内部に持つ各バージョンのファイル名を付ける (図 2.2)。

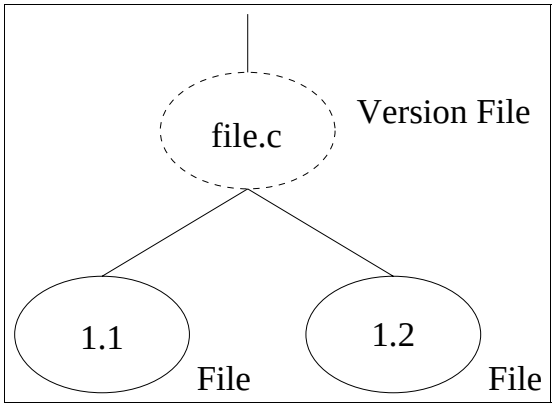

図 2.2: 3-D File System のディレクトリ構造の例

ユーザが Version File にアクセスすると、システムが適切なバージョン番号を選択し、ユーザはその内容をアクセスする事になる。また、特にバージョン番号を指定したい場合には、 “Version File”/“バージョン番号” という指定でアクセスできる。

Viewpath

Viewpath は、ディレクトリの重ね合わせを実現する概念である。あるディレクトリがアクセスされた時に、 Viewpath に指定された複数のディレクトリ以下の内容も同時に参照できる。 Viewpath に指定されたディレクトリ間には優先順位があり、同一の名前のファイルがある場合には優先順位が高い方が選択される。

開発中で未公開のファイルは、通常の利用者に見えないようにするという機能が導入されているが、ライブラリによって実現されているためコマンドの再コンパイルが必要であること、基本的に開発者が取り扱うファイル以外は Read Only として扱われるため利用に制約が生じる可能性がある。

2.1.3 Translucent File Service (TFS)

3D ファイルシステムと同様に、ソフトウェアのバージョン管理に用いられるファイルシステムで NFS マウント機能の拡張により実現されている [117]。

TFS では、ディレクトリを多重にマウントすることによってできる階層の上下関係によって可視性を制御する方法を用いている。通常の NFS マウントの場合、マウントするとその下になるディレクトリやファイルは見えなくなってしまう。しかし、TFS のマウントではマウントでできる階層は透明なので下のファイルは透けて見える。同名のファイルが複数の階層に存在する場合、下のファイルはそれより上にあるファイルによって隠蔽され、それらのファイルのうち最上の階層にあるファイルが見える (Figure 2.3参照)。

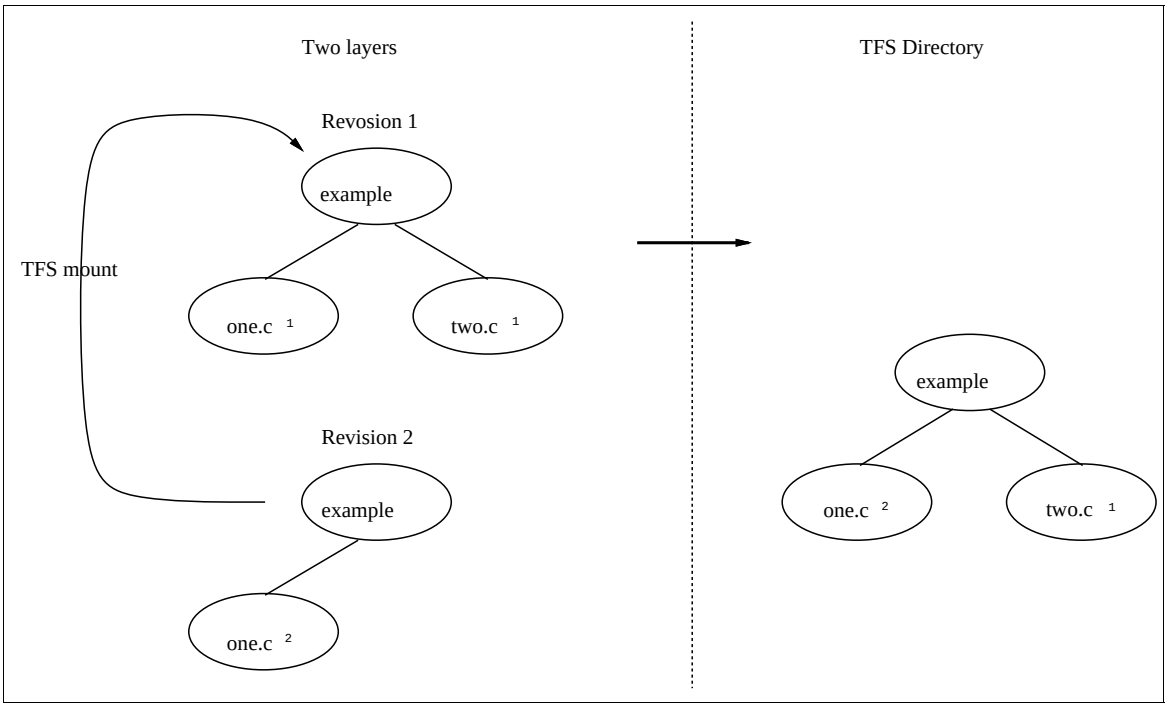

図 2.3: 2 階層の TFS ディレクトリ

TFS では、最も上の階層だけ書き込み可能で、その他の階層はすべて読み込み専用である (Figure 2.4。読み込み専用の階層にあるファイルを変更しようすると、書き込み可能な階層 (最上の階層) に複製された後、変更される。

TFS では、読み込み専用の階層にあるファイルを削除することはできない。もし、削除する必要がある時は、*whiteout* というエントリ最上の階層に作成することによって削除の代替とする。透明な階層に修正液を塗って下を隠すのと同じである。

TFS では、ディレクトリを読んだり (readdir) ファイルを探したり (lookup) する場合、それぞれの階層を最前から最後へと順々に探索しなければならない。その際、途中で見つかったファイルに関してはそれより下の階層を探索する必要はないし、*whiteout* のエントリがあれば、その時点でそのファイルは存在しないことにして良い。

しかしながら、ディレクトリの階層の数が多くなるとやはり探索のオーバヘッドが大きくなる。そこで一度 readdir や lookup の操作が行われると、その結果は最前の階層に backfiles というファイルにキャッシュとして保存される。最前の階層以外に backfiles が存在する場合、それは readdir や lookup の操作の時に参照にされ、それより下の階層を探索する必要がなくなる。

ソフトウェアのバージョン管理と共に、CD-ROM などの Read Only のデバイス上でコンパイル作業などをするためにも用いられる。NFS のマウントのメカニズムを用いて実現されており応用範囲も広いが、重ねられたディレクトリ間の関係がファイルシステム中に保持されていないという問題点を持つ。

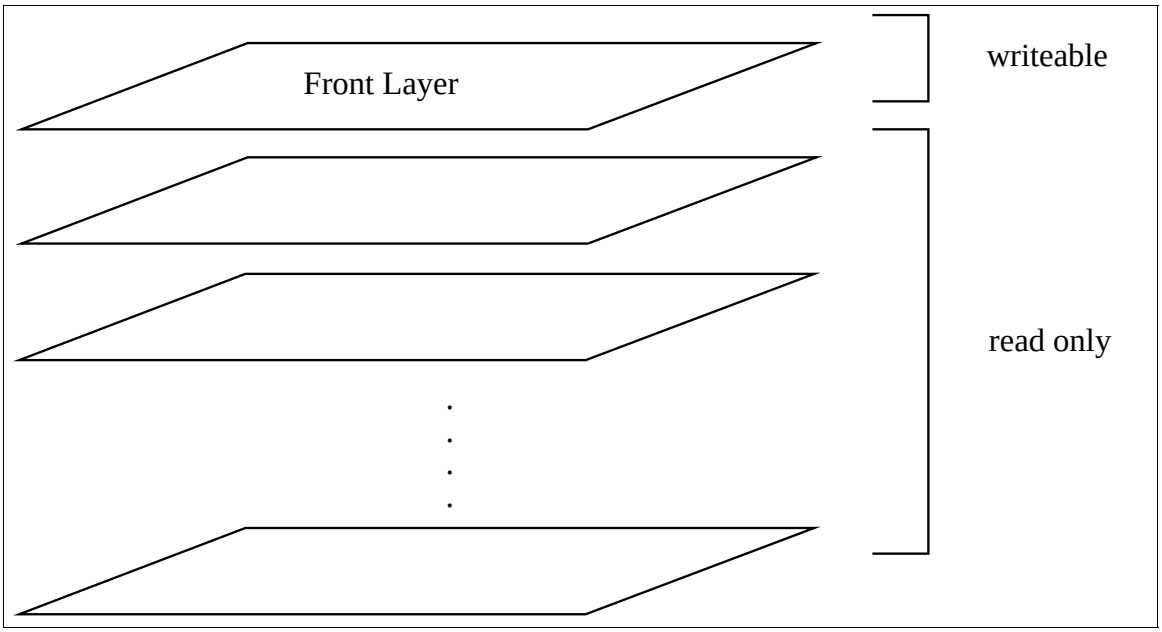

図 2.4: 階層の読み書き

2.1.4 AtFS(Attributed File System)

Attributed File System(AtFS) は、ソフトウエアの共同開発におけるバージョン管理に主眼を置いた可視性制御システムである [118]。ShapeTools と呼ばれる豊富なバージョン管理コマンド群によって、“AtFS” というディレクトリ内のファイルを管理し、ユーザに疑似的なファイルシステムを提供することを可能にしている。

基本的に次の三つのコマンドを使用するだけで、バージョン管理を行うことが可能である。

- save ファイルのセーブ
- retrv 指定されたバージョンの取り出し
- vl ファイルのリスト表示

この他、バージョン管理に必要なオペレーションを行うための豊富なコマンドが用意されている。

また、 AtFS では、ファイル名に以下のような形でバージョン番号を付加してアクセスする事で、バージョン番号を特定する。

filename.txt[1.1]

例として、 AtFS 用の ls コマンドである “vl” コマンドの出力例を以下に示す。

```
-rw-r--r-- b koni          36450 Oct 14 12:57 machines.h
-rw-r--r-- s koni          36450 Oct  2 16:35 xyzzy.c[1.0]
-rw-r--r-- p koni          36494 Oct  3 20:41 xyzzy.c[1.2]
```

この形式の情報のうち、通常の ls コマンドとの違いは、以下のような部分である。

- リビジョンの状態 (b=busy, s=saved, p=proposed, P=published, a=accessed, f=frozen) の頭文字がファイルパーミッションの次の部分に現れる
- ファイル名にバージョン番号を付加した形で表示する

この vl コマンドの例のように、豊富なコマンド群によって疑似的なファイルシステムをユーザに提供するという特徴がある。

複数のユーザコマンドで実装されているので、個人でシステムの導入を行えるが、このシステムに対応していない標準的な UNIX コマンドからの利用の場合はファイルの指定方法を変更する必要があるなど、一貫したファイルアクセス方法をとることが難しい。また、バージョン管理に主眼を置いているので、その他の用途にはあまり向いていないと考えられる。

2.1.5 View Control File System

設定の柔軟性、ファイルを見え無くする機構、ファイル実体とファイル名の関係の管理機構などの問題を解決するために提案された方法が、ここで提案する可視性制御機構 **View Control File System (VCFS)** である。ここでは、以下のような機能を提供している。

- 柔軟な設定機能
  アーキテクチャやオペレーティングシステムのバージョンによる切り替えとユーザやグループによる切り替えを提供する。また、いずれの場合でもファイルに対する読み出しおよび書き込みの両方のアクセスを可能にする。
- ファイルを見え無くするメカニズム
  条件に対応するファイル実体が存在しない場合には、ファイルそのものの存在を隠す。
- ファイル実体とファイル名の関係
  ファイル実体とファイル名の関係を明確にするため、対応するファイル実体を集めて管理する機能を提供する。

これらの機構を実現するためアーカイブディレクトリと呼ばれるディレクトリを導入し、複数のファイル実体とファイル名の管理を行なうようにしている。

対応する複数のファイル実体は、**アーカイブディレクトリ**と呼ばれるディレクトリに集められ、ファイル名はアーカイブディレクトリの名前として扱われる。図 2.5の例では、ファイル名として csh という名前がアーカイブディレクトリに与えられており、それぞれのファイル実体は sparc、mips、powerpc の 3 つのファイルが与えられている。これらファイル実体のファイル名は、実際のファイル名ではなく条件にしたがってファイル実体を検索するときのキーとして扱われる。例えば、キーとして `mips` が選択された場合、この csh という

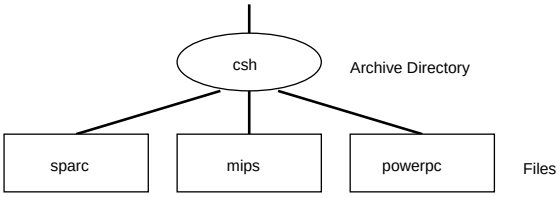

図 2.5: アーカイブディレクトリ

アーカイブディレクトリに csh という名前でアクセスすると、その下に置かれたファイル実体のうち mips という名前の与えられたものが選択されアクセスされることになる。

なお、実装上アーカイブディレクトリであるか否かの判断は、

ディレクトリでありかつ **SetUID bit** が設定されている

という条件を用いている。このようにすることにより、システムの変更を最小限ですませるようにしている。またアーカイブディレクトリ下のファイルをアクセスしたい場合には、... という表記を用いることにより可能にしている。例えば、前述の例では csh/... という表記によってアーカイブディレクトリ内のファイル実体のリストを表示することができるようになる。

```
% ls csh/...
mips     powerpc  sparc
%
```

現バージョンでの実現では、NFS プロトコルを用いデーモンを介してファイルシステムの動作を制御する方法を用いている。本方法では、コマンドの再コンパイルが不要であるとともに、変更の影響は特定の部分木に制限することが可能となる。ただし、処理のオーバヘッドが大きいという問題点を持つ。

ファイル実体の切り替えの条件となるキーの選択は、アーカイブディレクトリ内に vcfsd.conf (図 2.6参照) というファイルがある場合には、その設定にしたがって、そうで無い場合には mount 時に指定したキーにしたがって行うようになっている。

```
# Example of vcfsd.conf
default = others  # デフォルトで用いるキー
mine: suna, ray   # ユーザ suna と ray に
                  # 対するキー

:group            # グループ設定開始を示す
                  # 予約語
sunalab: SUNA_LAB # グループ SUNA_LAB に
                  # 対するキー
```

図 2.6: vcfsd.conf の例

第 3 章

応用

以下に、可視性制御機構の応用例を示す。

3.1 実行環境

マウントオプションにアーキテクチャやオペレーティングシステムのバージョンの違いを示すキーを与えることにより、自分の環境にあわせた実行モジュールを選択することが可能となる。また、コンパイル作業においても、異なるシステムの`*.o`ファイルが混在することがなくなり、従来の共有ファイルシステムで発生していたトラブルが削減できる。

3.2 バージョン管理

`vcfsd.conf`に指定されたユーザやグループによるキーの選択を行うことにより、共同作業環境におけるバージョン管理機能が実現できる。これにより同一ソースツリーの中で、異なるユーザが独自の変更を行いながらソフトウェアの共同開発ができるようになる。現在 RCS に組み込んだシステムを開発中である [119]。

3.3 Personal File System

現在、可搬型計算機がネットワークから切断されている状態でも保持しているキャッシュを用いて作業を継続できるようにするためのシステム Personal File System (PFS)[120] を開発している。この際、ネットワークから切断中に、可搬型計算機で行ったファイルの変更とサーバ側で発生した変更が衝突する場合がある。このとき、衝突を利用者に知らせると共に、衝突したファイルをアーカイブディレクトリに同一名で格納ができるようにしている。

3.4　ファイル名と格納形式の自動変換

分散ファイルシステムでは、サーバ側から提供されるファイルの格納形式がそのままクライアント側で利用できるとは限らないことが多い。例えば、格納されている漢字コードと、クライアント側で利用するアプリケーションプログラムが対応する漢字コードが異なる場合、「変換」を行なわなければならなくなる。このような場合に、変換されたファイルを仮想的な実体ファイルとして提供することにより、暗黙にファイルの格納形式の変換を行なう機構を実現することが可能である。以降では、こうした機構の実現について述べる。

第 4 章

可視性制御機構の拡張

4.1 拡張機能

動的にフィルタを組み込み、ファイルの read/write において自動的にファイルの格納形式を変換する機構を実現する。これは、アーカイブディレクトリにおいて、ファイル名 (拡張子) によって格納形式を指定し、それに基づき適当なフィルタを介してファイルへの read/write を実行する機能を実現するものである。これにより、以下のような機能を提供することができるようになる。

- 文字 (漢字) コードの自動変換
  格納されている文字コードにかかわらず、拡張子を適当に指定することで、アプリケーションが利用可能な文字コードに自動的に変換してアクセスできるようにする。

- 画像ファイルフォーマット
  文字コードの場合と同様に、アプリケーションが利用可能な画像ファイルフォーマットに変換してファイルを提供する。これにより、格納されたファイル形式にアプリケーションが対応していない場合でも、そのファイルを利用可能とすることができるようになる。

- 圧縮ファイルシステム
  圧縮されて格納されているファイルでも、通常のアプリケーションからのアクセスは自動的に展開されて行なわれ、圧縮ファイルの拡張子を指定することで圧縮されたままの形式でファイルのコピーが行なえるということができるようになる。例えば、「more FILE」とすることで、FILE の内容を表示することができ、「cp FILE.gz NEWFILE.gz」とすることで圧縮形式のままコピーが可能となる。

これらの機能を実現するためには、ユーザプログラムは、

- 目的のファイル名

- 要求するデータの形式

という、二つの情報を指定して、ファイルアクセスを行う必要がある。しかしこの為にファイル操作を行うシステムコールの仕様を変更することは望ましくない。

そこで、ファイル名に、そのフォーマット名を表す識別子を拡張子として付加して、それらを一まとめにして、一つのオブジェクトとして扱うようにしている。つまり、オブジェクトの形式は以下のようになる。

[ファイル名].[フォーマット名]

拡張子としては以下のようなものが考えられる。

- 圧縮されているか非圧縮か、といった違い
- 文字コードの違い (漢字コードで言えば jis, sjis, euc など)
- 画像フォーマットの違い (gif, jpeg, tiff など)

このファイル名と拡張子を併せた表記は、通常良く利用される表記方法であるため、ユーザは違和感をなく利用できるようになる。以下に、いくつか例をあげる。

`kanji_code.jis`	内部の漢字コードは JIS である。
`kanji_code.sjis`	上記ファイルと情報としての内容は同じであるが、漢字コードが SHIFT JIS である
`filename.gz`	GNU gzip で圧縮されているファイル

ユーザがこの形式のオブジェクトをファイルアクセスの引数として用いる事で、目的とする情報 (ファイル) を目的の形式 (拡張子で表されるフォーマット) で、取り出す事が可能となる機構が実現できる。

4.2　実現

ファイルアクセスの引数として利用されたオブジェクトから、システム側では、目的とされているファイル名と、要求されているフォーマット名とを識別する必要がある。そして必要であれば、要求されているフォーマットへ情報を変換しなければならない。

この時に用いるフィルタプログラムは、ユーザが動的に変更可能とすることで、より柔軟なファイルシステムを構築する事が可能となる。

そこで、アーカイブディレクトリの機構のうち、

- パス名の変換の仕組み
- ユーザ毎の可視性制御を行う時に利用する設定ファイル (vcfsd.conf)

を拡張して用いる事で、この機構を実現する。
具体的な仕様は以下の通りである。

- アーカイブディレクトリ名には拡張子なしのファイル名を用いる
  アーカイブディレクトリ自身の名前は、内部に置いた実体となるファイル群を表す名前を付けるので、本実装では拡張子を付けない純粋な情報としての名前を用いることにした
- ファイル名とフォーマット名に分離はサーバ上で行なう
  ファイルアクセスの引数として渡された情報を、 サーバプログラム 内部でファイル名とフォーマット名に分離して使用する
- vcfsd.conf ファイルにフィルタを定義する
  可視性制御ファイルシステム vcfsd によって、ファイルアクセス毎に vcfsd.conf ファイルが更新されたかチェックされるので、ユーザが動的にフィルタプログラムを変更することが可能となる
- 呼び出し側には定義されているフィルタを通したデータを返す
  フィルタプログラムによるフォーマット変換をシステム内部に完全に隠蔽する事で、ユーザプログラム側で特別な処理を行う必要がない

4.3 フィルタプログラムの定義

ファイルアクセス時に動的に起動されるフィルタプログラムの仕様、そして vcfsd.conf ファイルにフィルタを記述する方法に付いて定義する。

4.3.1 フィルタプログラムの仕様の定義

フィルタプログラムの仕様を以下のように定義する (図 4.1)。

- 標準入力からデータを読み込む
- 内部処理を施した結果を標準出力に書き出す

このフィルタプログラムの仕様は、 UNIX の標準的なフィルタプログラムの仕様であり、UNIX システムで多用されるフィルタプログラムを利用できるメリットがある。

4.3.2 フィルタプログラムの記述方式の定義

ユーザによるフィルタプログラムの定義はアーカイブディレクトリ内の vcfsd.conf ファイルに記述する。 vcfsd.conf ファイルへ記述する時の仕様を以下のように決める。

図 4.1: フィルタプログラムの仕様

- フォーマット A(拡張子 formA と表す) からフォーマット B(同 formB) への変換時に使用するフィルタプログラムを、以下のように定義する。
  .formA.formB: |filterprog_for_AtoB
- 特に拡張子なしの場合は以下のように記述する。
  .formA.: |filterprog_Ato
  ..formB: |filterprog_toB
- 複数のフィルタプログラムを連続して使用する場合には以下のように記述する。
  .formA.formB: |filterprog1 | filterprog2 | ... | filterprogN

以下に、簡単にフィルタプログラムの定義例を示す。

- .gif.jpg: |gif2jpg
  gif 形式のデータを jpg 形式で読み出したい場合
- .gz.: |zcat
  gzip で圧縮されたファイルを、圧縮されていない状態で読み出したい場合
- ..gz: |gzip -c
  データを gzip で圧縮した状態で読み出したい場合

4.4　フィルタプログラムを介した読み込み

読み込み時には、フィルタプログラムで処理された内容のファイルをディスク上に展開するのではなくて、フィルタを介した出力をメモリ上に読み込むことで処理の高速化をはかる。一般にディスクアクセスは遅いので、メモリ上で展開する事で処理を高速化できると期待できる。また、フィルタを通した読み込みはプロセスを fork して、ファイルの内容に対する動作とフィルタを介した変換とを並列に処理する。

また、 nfs 上でのファイルの読み書きは 1024 以上の 2 の冪乗で表されるサイズ (例えば 8192 バイト) のデータを一度に扱う。そこで、このブロックサイズ分のバッファを用意し、フィルタを介してメモリ上に読み込んだデータをキャッシュする事で、読み込んだデータに対する lseek 等のファイル操作をメモリ上で行う機構を実装する。

4.5 フィルタプログラムを介した書き込み

書き込みが行われる時には、大きく分けて次の二つの場合が考えられる。

(1) 最終的な形式が決まっている場合
例えばファイルを圧縮しておくような場合

(2) データの情報としては同じであるのでどんな形式でもいい場合
例えば漢字コードの違いなどの場合 (sjis, euc, jis など)

この二つを区別するために、アーカイブディレクトリ内に置く実体となるファイルについて、(1) の場合は拡張子でその形式を明示し、(2) の場合には拡張子による形式の明示は行わずにファイル名のみとして、区別する事にする。

この様に仕様を決定する事で、書き込みが行われる時に、以下の三通りの場合に更に細かく分けられる。

(1a) 最終的な形式で書き出す場合 (拡張子付きで書き込まれた場合)
→ そのままデータを書き込むことが可能

(1b) 最終的な形式にするためにはフィルタを通す必要がある場合 (拡張子なしで書き込みが行われた場合)

(2) 形式にこだわらずに書き出す場合 → そのままデータを書き込む事が可能

このため、書き込み時にフィルタプログラムを利用する必要があるのは (1b) の場合のみとなる。(1b) の場合でも、書き込み時にフィルタプログラムの処理が終るまで待たされる様な事は避けたい。

そこで、(1b) の場合、まず拡張子なしのファイル名でそのままデータを書き込む。この時、可視性制御によってユーザ側には新しく書き込まれた方のデータを見せる。そして書き込みが終了した時点で、プロセスを fork してフィルタプログラムを独立に実行し、今書き込んだデータを処理する事によって、最終的な形式に遅延変換を行う。

こうすることで、ユーザはファイル書き込みにおいてフィルタ処理に時間がかかるような場合でも待たされる事はない。また、この方法は書き込むデータを一度確実にファイルに書き出してから処理するので、データの安全も確保できる。

第 5 章

実装

前章で述べた仕様に基づき実装を行った。現在、NetBSD 1.1(based on BSD4.4), BSD/OS 2.0.1(based on BSD4.4) 上での動作を確認している。

この章では、機能拡張を行った vcfsd のシステム構成と、特に重要な RPC 処理関数について述べる。

5.1 システムの構成

5.1.1 動的にフィルタを介したファイルアクセス

可視性制御ファイルシステム vcfsd に対して機能拡張を行った。このシステムでは、動的にフィルタを介したファイルアクセスは、以下の図 5.1 の様に行われる。

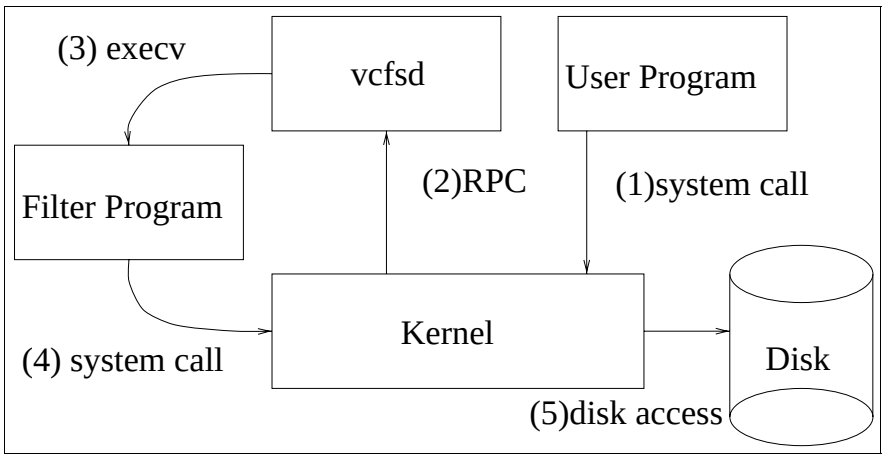

図 5.1: 動的にフィルタを介したファイルアクセス

目的とするファイル名を filename 、要求されるフォーマット名を ext と表すと、フィルタを介したファイルアクセスは以下のように行われる。図 5.1 と連動する形で説明を行っていくことにする。

(1) ユーザプログラムからのファイル呼び出し
この時、ユーザプログラムは filename.ext という形式をファイルアクセスのシステムコールの引数として用いて、ファイルをアクセスする。

(2) カーネルからの RPC 呼び出し
カーネルは vcfsd で管理されているディレクトリ以下のファイルのアクセスに対しては、 RPC の関数呼び出しの形で vcfsd へ処理を任せる。この時の関数の引数として filename.ext が利用される。

(3) パス変換とフィルタプログラムの呼び出し
vcfsd は受け取った filename.ext という引数を、 filename と ext に分離し、 filename で表されるファイルの内容を ext で表される形式に変換するフィルタプログラムを呼び出す。この時、フィルタプログラムの標準入力を filename に、標準出力を vcfsd で受け取るように設定してからフィルタプログラムを実行する。

(4) フィルタプログラムによるファイルの読み込み
フィルタプログラムは標準入力からファイルを読み込む。この時、フィルタプログラムでは、カーネルに対して標準入力を読み込むシステムコールを実行する。

(5) カーネルによるディスクアクセス
カーネルによって実際のディスクアクセスが行われる。

この様にして、フィルタを介したファイルを読みだし、その結果はこの逆の順序でユーザプログラムに返される。

フォーマット変換はファイルシステムの内部で行われ完全に隠蔽されるので、結果としてユーザプログラムでは、要求した情報を受け取ることが可能となる。

5.1.2 ディレクトリ構成

ディレクトリ構成の例は、図 5.2 の様になる。この例を用いてフィルタを介したファイルアクセスについて説明する。

- アーカイブディレクトリ名
アーカイブディレクトリの名前は拡張子なしの kanji という、内部に置いた情報を表す名前を用いる

- アーカイブディレクトリ内のファイル
アーカイブディレクトリ内には、元の情報となるファイルである kanji というファイルと、可視性制御の設定ファイルである vcfsd.conf ファイルの二つだけが置かれる

- フィルタの定義
  フィルタの定義は、元となる kanji と言うファイル、そのファイルを euc や sjis に変換する時に利用するフィルタ名を記述する

- ユーザプログラムによる kanji へのファイルアクセス
  ユーザプログラムが kanji というアーカイブディレクトリへアクセスした場合は、フィルタを通さずにパス名の変換だけを行い、内部の kanji というファイルをアクセスする

- ユーザプログラムによる kanji.sjis へのアクセス
  ユーザプログラムが kanji.sjis というファイルをアクセスした場合、実際には kanji.sjis というファイルの実体は存在しない。ここで、 vcfsd によるパス名の変換が行われて、 kanji.sjis は kanji にフィルタ処理を施して得られるので、フィルタプログラムを呼び出した結果が返される。

以上のような形でアーカイブディレクトリ内のファイルに対して、フィルタプログラムを介したファイルアクセスが行われる。

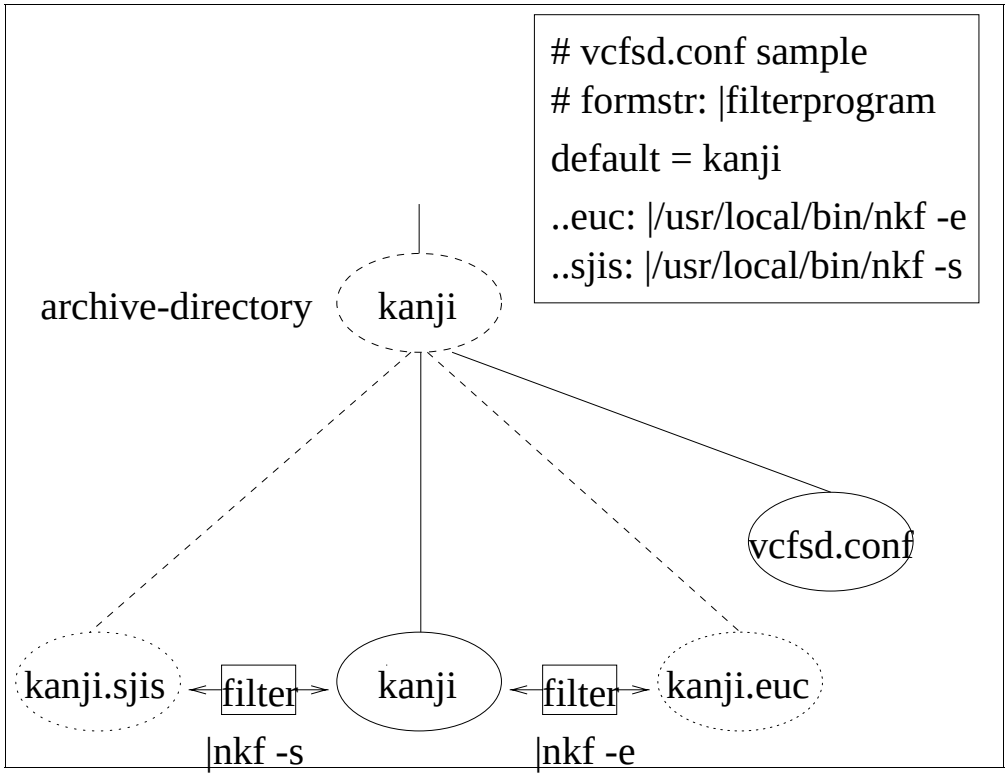

図 5.2: ディレクトリ構成の例

第 6 章

まとめ

ファイルの可視性制御は、広域分散環境で必要とされるさまざまな機能をファイルシステムに隠蔽することで、利用者からの利用を容易にすることを目的としている。

現在、システムの実装とともに、より具体的な応用について検討を行なっている。実用的な処理速度を実現するとともに、実際の利用形態においてどのような機能が必要なのかを絞り込んでいくことが今後必要になってくるであろう。