平成19年度　　卒業研究論文

Databaseを用いた検索機能を有する
ユーザレベルファイルシステムの構築

指導教官
齋藤 彰一 准教授

名古屋工業大学　情報工学科
平成16年度入学　16115059番

栗本 裕司

目 次

第1章

はじめに

近年ではストレージ技術の発展により，TB 単位の大容量のストレージが個人でも所有できるようになった．これにより，計算機のストレージには数十〜数百万ものファイルが存在できるようになった．それにともない，大量のファイルをわかりやすく管理する必要が出てきた．

ストレージ上のファイルを管理するために OS ではファイルシステムを提供している．一般的な OS(Windows や Linux) では木構造に従った階層的なファイルシステムを提供している．このようなファイルシステムはユーザにとって使いやすく，柔軟にファイルを分類することができるため，長年に渡り広く普及し利用されてきている．

しかし，この木構造に従った階層的なファイルシステムには「作成したファイルパスでしかアクセスできない」と言う欠点をもつ．この欠点はファイルの量が少ないときにはさして問題ではないのだが，ストレージ内に大量のファイルがあった場合この欠点によりファイルのアクセスが困難になってくる．

本論文では，木構造に従った階層的なファイルシステムではなく，ファイルに複数のキーワードを付けそのキーワードを検索することでファイルを特定するファイルシステムを提案する．また，提案手法をユーザレベルファイル構築ツールとオープンソースのリレーショナルデータベース管理システムを用いて実装した．

以下，2 章では現在良く使われている木構造に従ったファイルシステムについて説明し，3 章では，関連研究について述べ，4 章では提案手法について述べる．5 章で提案手法の実装と実装したファイルシステムの評価を行い，6 章で結論をまとめる．

第2章

木構造ファイルシステム

本章では，一般的な OS に使われている木構造に従ったファイルシステムの基本的な特徴と欠点を説明する．

2.1　木構造ファイルシステムの特徴

Windows や Linux などの一般的な OS に使われているファイルシステムはディレクトリ (フォルダ) と呼ばれる概念をもつ．ディレクトリというのは，共通の目的をもつファイルを整理・管理するためのものであり，ディレクトリの中にはファイルとディレクトリを入れることができる．このディレクトリを用いることにより，ディレクトリの中に別のディレクトリを作成し，またそのディレクトリの中に別のディレクトリを作成する，といった様に，階層的にディレクトリを作成でき，これらの階層を用いてファイルを分類することができる．

階層構造の例を，図 2.1 に示す．一般的な場合，ファイルシステムは図 2.1 のようなルート (/) と呼ばれるディレクトリを頂点とした階層構造になる．このルートディレクトリの下に階層的にディレクトリを意味や用途によって分類しながらファイルを格納していく．

またこの階層構造の中でファイル名を指定する場合次のの 2 種類の方法がある．

- ルートディレクトリを基点として目的のファイルまでのすべての道筋を記述する絶対パス方式

- 現在いるディレクトリを基点として目的のファイルまでのすべての道筋を記述する相対パス方式

したがって，図 2.1 において”検討会資料”ディレクトリ内にいたときに，”foo.txt”にアクセスするには，絶対パス方式では”/検討会資料/4 月/foo.txt”となり，相対パス方式では，”4 月/foo.txt”となる．図 2.1 のような構造は，ルートを根とした木のような構造をしているため木構造と呼ばれる．

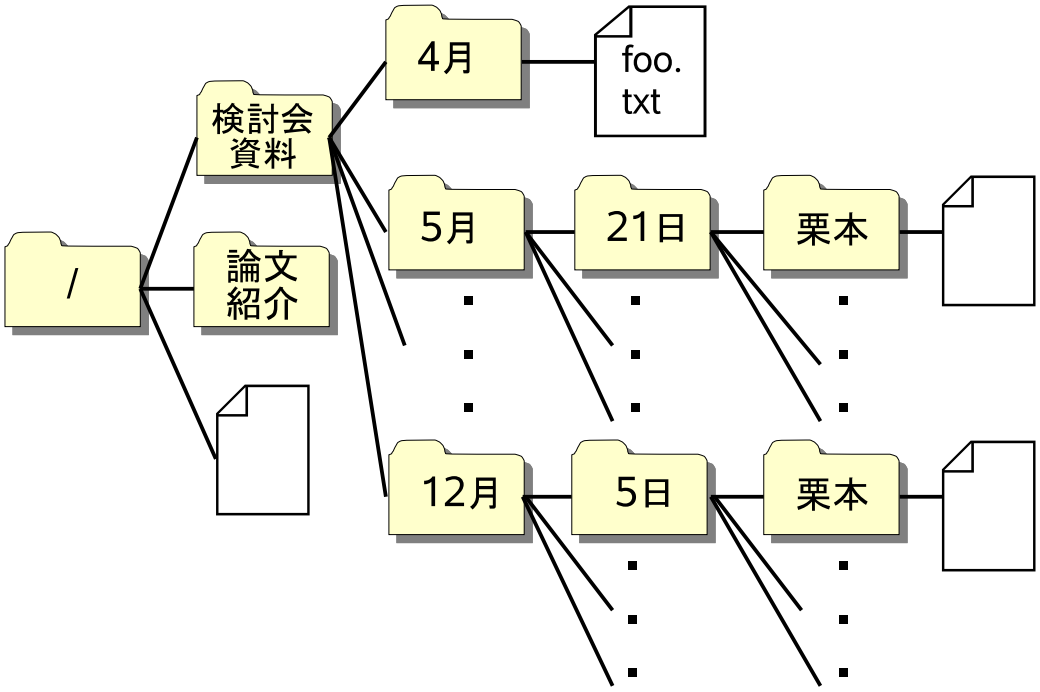

図 2.1: 階層構造の例

このような木構造ファイルシステムは階層構造をもたないフラットな構造のものと比較すると，次のような利点がある．

- ユーザにとって使いやすく柔軟なファイル分類機能を提供できる
- 住所や組織などの元々木構造のものを適用するのに有効である

2.2 木構造ファイルシステムの欠点

木構造ファイルシステムには，作成したファイルパス 1 つのみでしかファイルにアクセスできないという欠点がある．例えば，ファイルを細かく分類するためにディレクトリの階層を深くした場合，あるファイルにアクセスするには次のようなパスでアクセスしなければならない．

”/検討会資料/5 月/21 日/栗本/kurimoto-ken-20070521.ppt”

このような長いパスはファイルを分類する上ではわかりやすく，また，ある階層以下に同名のファイルが複数存在しているときには有効である．しかし，このファイルをアクセスするにはこの長いファイルパス 1 つのみでしかアクセスできないため非常に手間がかかる．なので，このファイルパスをもっと短いパス，例えば，

”/検討会資料/栗本/kurimoto-ken-20070521.ppt”

というパスでもアクセスしたいという要求がある．

次に，様々な場所に散在している項目を集約するのにも非常に困難である．図 2.1 を参考にすると，例えば図 2.1 に散在するディレクトリ”栗本”以下のファイルを集めたいというときには，ルートディレクトリから木全体をたどり，そこに”栗本”というディレクトリが存在しているかどうかを探索しなければならず非常に手間がかかる．

また，木構造ファイルシステムではディレクトリの階層の順序を状況に応じて動的に入れ換えることができない．例えば，”/検討会資料/4 月/foo.txt”というファイルパスをもつファイルにアクセスするにはこのパスでしかアクセスできないが，使用する状況によっては”/4 月/検討会資料/foo.txt”というようなパスでアクセスしたいという要求もある．

2.3 シンボリックリンク

UNIX 系 OS では前節で挙げたような問題を緩和するためにシンボリックリンクという機能をそなえている．シンボリックリンクとは，あるファイルやディレクトリに

別の名前 (パス名) を与え，ユーザやアプリケーションがその名前を元のファイルと同様に扱えるようにできる機能のことである．

このシンボリックリンクを用いることで，前節で挙げた長いファイルパスの場合にはそのファイルパスに対して”/検討会資料/栗本/kurimoto-ken-20070521.ppt”というシンボリックリンクを作成することによって長いファイルパスも短いパスでアクセスすることが可能となる．またファイルを集約する場合にも，ファイルを作成したときにあらかじめその項目 (前節の例では”栗本”) のディレクトリを作成しておきそのディレクトリ内に作成したファイルのシンボリックリンクを作成すれば木全体を検索しなくてすむ．

しかし，このシンボリックリンクを作成する方法は，シンボリックリンクを作成する必要があるファイルが少ないときには非常に有効だが，作成する必要があるファイルが増えれば増えるほどシンボリックリンクを作成するのが困難になってしまう．

第3章

関連研究

前章で挙げたように従来の木構造ファイルシステムには，ファイル数が増大することでファイルを特定することが非常に困難になってしまうという問題があった．

現在この問題を解決するために，主に,「逐次検索を行わない検索システムの作成」,「木構造ファイルシステムとは異なるファイルシステムの構築」の 2 種類のシステムが研究されている．

本章では，逐次検索を行わない検索システムの作成する方式の研究として「デスクトップサーチ」と「全文検索エンジンを利用したファイルシステムの名前空間拡張 [1]」を，木構造ファイルシステムとは異なるファイルシステムの構築の研究として「SetNS[2]」について紹介する．

3.1 デスクトップサーチ

デスクトップサーチとは，計算機内のデータや電子メールのメッセージを素早く簡単に検索するためのツールである．デスクトップサーチは，計算機内に保存されているファイル内のテキスト情報を事前に解析し，データベース化をおこなう．このようにデータベース化をおこなうことで，ファイル名だけではなく，ファイルの内容についても検索可能となる．さらに，ファイルの検索に，ファイル自体を逐次検索するのではなく，データベースに登録された内容を検索することにより，高速に検索結果を得ることができる．

主なデスクトップサーチとして，Windows Desktop Search[3]，Google Desktop[4] が

良く知られている．これらのデスクトップサーチは文書ファイルやメールの内容だけではなく，マルチメディアや画像ファイルに付けられている情報でも検索することができる．

3.2 全文検索エンジンを利用したファイルシステムの名前空間拡張

論文 [1] では，全文検索エンジンを利用したファイルシステムの名前空間拡張方法について書かれている．この論文では従来の木構造ファイルシステムに全文検索エンジンを統合し，ファイル名だけでなくファイル内容についても検索することを可能としている．

このシステムでの検索方法は特殊な文字列をともなうディレクトリを使用する．この特殊な文字列を用いることにより，完全一致や前方一致等の指定や and，or の演算子の指定を可能としている．そしてその特殊な文字列をともなうディレクトリ名を検索記号と見なし，検索式を生成し全文検索エンジンによる検索を行う．このようにして検索した結果はディレクトリ内にシンボリックリンクとして表現される．

また，このシステムは拡張された名前空間も木構造に準じているため，ファイルアクセス API を用いるアプリケーションすべてに，変更を加えることなく，検索機能を提供することができる．

3.3 SetNS

3.3.1 概要

SetNS(Set Name Service)[2] は，木構造ファイルシステムとは異なるファイルシステムを構築している．SetNS では，ファイルパスが記号の集合として扱われる．この SetNs における記号とは区切り文字以外の文字からなる文字列である．

もし，

”/検討会資料/5 月/21 日/栗本/kurimoto-ken-20070521.ppt”

というパスで指定できるファイルが存在した場合，SetNS では，

検討会資料,5 月,21 日, 栗本,kurimoto-ken-20070521.ppt

というファイルパスに現れてくるディレクトリ名，ファイル名を記号とみなす．SetNS におけるファイルパス，記号を区切り文字’,’ で結合したものとして管理される．この SetNS ではファイルパスを記号の集合として扱うため，記号の並び順に意味はない．

3.3.2 利用例

SetNS で次のファイルが存在した場合を例に述べる．

検討会資料,5 月,21 日, 栗本,kurimoto-ken-20070521.ppt
検討会資料,6 月,18 日, 栗本,kurimoto-ken-20070618.ppt
検討会資料,6 月,25 日, 高木,takagi-ken-20070625.pptx

ルートディレクトリでは、すべてのファイルを構成しているすべての記号が表示される．ファイルを特定することができる記号 (例：5 月，高木など) であればその記号はファイルとして表示され，ファイルを特定することできない記号 (例：検討会資料，6 月など) は，仮想ディレクトリとしてみなされる．仮想ディレクトリとは，実体が無いディレクトリのことである．仮想ディレクトリ内には，その記号が含まれているファイルを構成している記号が表示される．ただし，仮想ディレクトリに使われた記号は除かれる．例えば，仮想ディレクトリ”6 月”の中には 6 月という記号を含んだファイルを構成する記号のうち 6 月を除いたものが表示される (図 3.1 参照)．

SetNS でファイルにアクセスするには，その仮想ディレクトリにいる時点でファイルとして表示されている記号を指定すればよい．例えば，図 3.1 で示した仮想ディレクトリ”6 月”内で”kurimoto-ken-20070618.ppt”にアクセスする場合には，”kurimoto-ken-20070618.ppt”の他に，”18 日”，”栗本”といった記号でもそのファイルにアクセスすることができる．

```
%ls -l 6月/
-rw-r--r-- 1 kurimoto matlab 385664  1月10日 10:07 18日
-rw-r--r-- 1 kurimoto matlab 321499  1月10日 10:03 25日
-rw-r--r-- 1 kurimoto matlab 385664  1月10日 10:07 kurimoto-ken-20070618.ppt
-rw-r--r-- 1 kurimoto matlab 321499  1月10日 10:03 takagi-ken-20070625.pptx
-rw-r--r-- 1 kurimoto matlab 385664  1月10日 10:07 栗本
drwxr-xr-x 3 kurimoto matlab  24576  1月23日 12:48 検討会資料/
-rw-r--r-- 1 kurimoto matlab 321499  1月10日 10:03 高木
```

図 3.1: SetNS の利用例

3.3.3 特徴

SetNS はファイルパスを記号の集合として扱うため，木構造とは違い作成したファイルパスのみでしかアクセスできないという制限はない．これにより，SetNS は 3 つの特徴がある．

1 つ目に，ファイルにアクセスする際に長いファイルパスを短いパスで指定できるという点がある．これは前節で示したように，1 つの記号でそのファイルを特定することができた場合には，記号はそのファイルを示すことになるので，前節の例をとると，”kurimoto-ken-20070618.ppt”にアクセスするには，木構造ファイルシステムでは，”/検討会資料/5 月/21 日/栗本/kurimoto-ken-20070521.ppt” という長いパスでもこのディレクトリをたどってしかアクセスできない．しかし，SetNS ではルートディレクトリ上でも，”/kurimoto-ken-20070618.ppt”，”/5 月”，”/21 日”だけでファイルにアクセスできるので，わざわざ長いパスでアクセスする必要がない．

2 つ目に，様々な場所に散在している項目を集約するのが非常に簡単にできることである．例えば，図 2.1 の構造においてディレクトリ”栗本”以下のファイルを集める場合，木構造ファイルシステムの場合には散在しているディレクトリを探すのには木全体をたどる必要があった．しかし，SetNS では仮想ディレクトリ”栗本”に移動することによって簡単に”栗本”を含むファイルを集めることができる．

最後に，ファイルを指定する際に，ファイルの構成要素である記号を任意の順で指定可能という点である．木構造ファイルシステムでは使用状況に応じてディレクトリ

の階層の順序を動的に入れ換えることができなかった．しかし，SetNS では記号の並び順に意味はないので，仮想ディレクトリの階層の順序を動的に入れ換えることが可能である．

第4章

提案手法

本研究では，木構造ファイルシステムの問題点を解決するために，木構造ファイルシステムを用いないファイルシステムを提案する．本章では，本提案手法についてと使用法，そして利用における提案手法の制約の説明を行う．

4.1 概要

本提案手法は，従来のファイルシステムのようなディレクトリを用いた階層構造によってファイルを管理するのではなく，ディレクトリの変わりにファイルに複数のキーワードを付加することによって管理する．このようなファイル管理方法は関連研究である SetNS を踏襲した形になるが，本提案手法ではファイル名とキーワードを明確に区別した．これについては 4.2.1 項で詳しく説明する．このファイルとキーワードの関連付けはデータベースを用いる．

また，ファイルにアクセスするには，木構造ファイルシステムのようにディレクトリを辿っていくのではなく，ファイルに付加されているキーワードをデータベースに問い合わせて検索する，という形で行い，ファイルを特定できた時点でファイルにアクセスすることを可能とした．こうすることにより，木構造ファイルシステムでの作成したファイルパス 1 つのみでしかアクセスできないという問題が解消される．この問題が解消されるため，SetNS と同じように次の利点が得られる．

- 長いファイルパスを短いパスで指定できる

- 特定の項目を集約を容易にできる
- 1つのファイルを構成しているキーワードを任意の順番で指定可能である

検索結果には，検索に使用したキーワードが含まれているファイルとファイルに含まれているキーワードのうち検索に使用されたものを除いたものが表示される．

さらに，本提案手法は，既存アプリケーションのプログラムを変更することなく使用可能とした．API を新設することによって独自のシステムが構築可能である．しかし，本提案手法では既存のシステムとの互換性を重視し，既存プログラムの変更を不必要なものとした．これによって既存の appache(httpd) のようなサーバソフトウェアや samba システムなどが無変更で動作可能である．

本提案手法ではファイルシステム自体に検索機能を備えている．これにより，ファイル参照を行うアプリケーションに対して，プログラムの変更を一切することなく検索機能を提供することができる．デスクトップサーチではファイルを検索することはできるのだが，検索結果としてファイル名を出力するだけであり，その検索結果をアプリケーション側が使用することはできない．デスクトップサーチの検索結果を利用するためには，そのデスクトップサーチから提供される API を用い，アプリケーションにコードを追加しなければならなくなる．このコードの追加はユーザにとって大変な手間となってしまう．

論文 [1] のシステムと比較すると，既存のアプリケーションの変更無しに検索機能を提供するのは同じだが，検索結果にシンボリックリンクではなくファイルをそのまま表示するという点が異なっている．シンボリックリンクではファイルを編集することはできるが，ファイル本体のリネームや削除をすることはできない．だが，本提案手法ではファイル本体を表示するため、編集はもちろんファイルのリネームや削除をすることも可能である．

また，本提案手法は検索方法としてキーワードのみの指定で検索している．これは，機能性より視認性を重視したからである．論文 [1] のシステムの検索方法は特殊な文字列を用いて柔軟な検索方法をとることができる．しかし，特殊な文字列を用いる分パス名が長くなり視認性が悪くなる可能性がある．ただ，本提案手法は視認性を重視したため，OR 演算や前方一致などの柔軟な検索を行うことができない．

4.2 定義

4.2.1 ファイル名とキーワード

ファイル名とはファイルシステム中に保存されたファイルを特定するためにつける名前である．ファイル名は，ファイルを作成するときにユーザが付けるものである．本提案手法におけるキーワードとは，ファイルを 1 つに特定するための要素である．キーワードは，ファイルを作成するときにファイル名とともにユーザが付加する．本提案手法では，キーワードの組合せとファイル名で 1 つファイルを表現する．例えば，木構造ファイルシステムにおいて，次のファイルパスで表現できるファイルがあったとする．

”/検討会資料/5 月/21 日/栗本/kurimoto-ken-20070521.ppt”

このファイルを本提案手法で扱うと，ファイル名は”kurimoto-ken-20070521.ppt”だが，ファイルパスに存在するディレクトリ名それぞれがファイルを特定するためのキーワードとして扱われる．このようなファイル名とキーワード群はデータベースを用いて図 4.1 のように管理される．

ファイル名	キーワード
kurimoto-ken-20070618.ppt	検討会資料,6 月,18 日, 栗本
kurimoto-paper-20070925.ppt	論文紹介,9 月,25 日, 栗本
takagi-ken-20070501.pptx	検討会資料,5 月,1 日, 高木
index.html	検討会資料,6 月,11 日
index.html	検討会資料,6 月,25 日

図 4.1: ファイル名とキーワードの管理

さらに，本提案手法は，ファイル名とキーワードを明確に区別した．それによりファイルにアクセスする際には必ずファイル名でアクセスしなければならない．こうすることによって，アプリケーションが拡張子によってファイルの種別を判断できるようになる．関連研究である SetNS ではファイル名もキーワード (記号) の集合の 1 つの要素にとなっているため，ファイル名とキーワードの区別が非常に曖昧になっている．このよう

にすると，ファイルを構成しているキーワードでファイルアクセスができる反面，キーワードでアクセスできた場合，アクセスしたファイル名がわからないことになる．この場合，アプリケーション側ではどのような種類のファイルか判定できない可能性がある．例えば，図 4.1 にあるキーワード”検討会資料,6 月,11 日”をもった”index.html”を高機能エディタで開くときに，SetNS では”11 日”というキーワードを指定するだけで開くことが可能であるが，エディタ側は”11 日”という名前でファイルをテキスト編集モード開いてしまう．しかし，本来では”index.html”という HTML ファイルであるため，HTML 編集モードで開かれるべきである．なぜこのようになってしまうかというと，エディタはファイル名の拡張子を見て判断しているからである．本来ならば”index.html”というファイルを開くときはそのファイル名を指定するので，エディタはそのファイル名の拡張子を見てそれに適したモードで開くことができる．しかし，”11 日”という名前で”index.html”というファイルをエディタを開いた場合，そのファイルの種類が HTML ファイルであっても名前には拡張子が付いていないため，どのようなファイルかはエディタでは判断できなくなる．

4.2.2 仮想ディレクトリ

本提案手法では，登録されているファイルを見やすくするために仮想ディレクトリという概念を導入する．仮想ディレクトリというのは，ファイルシステムにおいてその場所に存在するように見えるのだが，実体がないディレクトリのことである．本提案手法のファイルシステムではキーワードが仮想ディレクトリとして判断される．本提案手法における仮想ディレクトリの内容は次のようになる．

- 仮想ディレクトリ名のキーワードを含んだファイル
- 仮想ディレクトリキーワードを除いたキーワード群 (これらのキーワードも仮想ディレクトリである)

本提案手法では，ユーザが仮想ディレクトリ内から明示的にファイルを削除したり，新たにファイルを作成することが可能である．

4.3 ファイル参照

4.1 節で説明したように本提案手法は，ファイルに付加されているキーワードを検索する形でファイルを参照する．図 4.1 に示されているファイルが存在したときのファイルの参照例を図 4.2 に示す．

```
%ls -l 検討会資料,6月/                                                    -- (1)
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:28 11日/
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:28 25日/
-rw-r--r-- 1 kurimoto matlab 385664  1月 10日 10:07 kurimoto-ken-20070618.ppt
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:28 栗本/
%ls -l 検討会資料,6月/11日/                                               -- (2)
-rw-r--r-- 1 kurimoto matlab    176  1月 23日 12:27 index.html
%cd 6月/検討会資料/                                                       -- (3)
%ls -l
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 11日/
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 25日/
-rw-r--r-- 1 kurimoto matlab 385664  1月 10日 10:07 kurimoto-ken-20070618.ppt
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 栗本/
%ls -l /栗本/                                                             -- (4)
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 18日/
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 25日/
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 6月/
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 9月/
-rw-r--r-- 1 kurimoto matlab 385664  1月 10日 10:07 kurimoto-ken-20070618.ppt
-rw-r--r-- 1 kurimoto matlab 827216  1月 10日 10:03 kurimoto-paper-20070925.ppt
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 検討会資料/
drwxr-xr-x 3 kurimoto matlab  24576  1月 23日 12:48 論文紹介/
```

図 4.2: ファイル参照の例

図 4.2 の (1) では”検討会資料,6 月”という仮想ディレクトリの内容を参照している．このように示したようにキーワードを”,”で繋げることで複数のキーワードを一度に指定することも可能である．この仮想ディレクトリの中にはキーワード”検討会資料”と”6 月”を含むファイルと，ファイルに含まれているキーワードのうち仮想ディレクトリに使用されたものを除いたものが表示される．このとき，”kurimoto-ken-20070618.ppt”にアクセスする場合，ファイルパスは”/検討会資料,6 月/kurimoto-ken-20070618.ppt”となりすべてのキーワードを指定することなく短いファイルパスでファイルにアクセスできることがわかる．

しかし，図 4.2 の (1) を見ると，キーワード”検討会資料”と”6 月”を含んでいるはずの”index.html”というファイルが表示されていない．これはキーワードの検索をおこなった結果，複数の同名ファイルが存在する場合そのファイルを表示しないように制限したためである．なぜ同じディレクトリに同じファイル名が複数存在してしまうかというと，従来の木構造ファイルシステムの場合，同じディレクトリに同じファイルを作成することはできない．しかし，本提案手法では，仮想ディレクトリの中はその仮想ディレクトリ名のキーワードを含んだものの検索結果が格納される．なので，同じキーワードをもった同じファイル名のものが存在した場合，その仮想ディレクトリに同じファイル名が存在することになる．もし，同じ仮想ディレクトリに同名ファイルが複数存在した時にそのファイル名を指定した場合，アプリーケーション側では同名ファイルの内のどれを指しているのか解釈することができない．よって，この問題を抑制するために，ファイルを表示しないように制限した．この制限により，同名ファイルの 1 つにアクセスする場合には，図 4.2 の (2) のように新たにそのファイルを特定するキーワードをユーザが新たに追加して検索しなおさなければならない．

図 4.2 の (3) では，cd コマンドにより仮想ディレクトリ”/6 月”の中の”検討会資料”という仮想ディレクトリに移動した後に，その仮想ディレクトリの内容を表示している．この仮想ディレクトリの中には，キーワード”6 月”と”検討会資料”を含んだものの検索結果が入っており，内容は図 4.2 の (1) のときと同じである．この結果よりキーワードの順番を入れ換えることが可能であることがわかる．

図 4.2 の (4) では，”栗本”という仮想ディレクトリの内容を表示している．この仮想ディレクトリの中にはキーワード”栗本”を含んだファイル全てが表示される．したがって，キーワード”栗本”を含んだファイルを集めたいという時にも容易に集めることが可能であることがわかる．

4.4 ファイルの作成

UNIX コマンドである touch コマンドを用いたファイルの作成例を次に示す．

% touch 検討会資料,6 月,11 日,foo.txt

このように，付加したいキーワードを”,”で区切ってつけていき，最後にファイル名をつけて作成する．このとき，本提案手法ではファイル名に必ず拡張子を付ける様に制限した．これは，キーワードとファイル名を明確に区別するために必要である．このようにファイルを作成すると，データベースにファイル名を”foo.txt”，キーワードを”検討会資料,6 月,11 日” として登録される．また，仮想ディレクトリ内でファイルを作成した場合は，ファイルパスに存在する仮想ディレクトリのキーワードも付加されて，データベースに登録される．

4.5 ファイルのリネームとコピー

本提案手法では，ファイルのリネームとコピーは，同一仮想ディレクトリ内でおこなうかどうかで意味が違ってくる．以下では，ファイルのリネーム，コピーについてそれぞれ説明する．

4.5.1 ファイルのリネーム

ファイルのリネームは，ファイル名を変更するだけではなく，ファイルの位置の変更もおこなう．本提案手法において，仮想ディレクトリ間の移動は，キーワードの付け替えを意味する．仮想ディレクトリは，その仮想ディレクトリ名のキーワードを含んだファイルを格納している．したがって，ファイルを仮想ディレクトリから移動することは，そのキーワードを含む場所から含まない場所へ移動することになる．つまり，ファイルは仮想ディレクトリをでることによって，ファイルからその仮想ディレクトリのキーワードを削除されることになる．また，ファイルを仮想ディレクトリに移動することは，そのキーワードを含まない場所から含む場所へ移動することになるので，ファイルにその仮想ディレクトリ名のキーワードを付加されることになる．

そのため，仮想ディレクトリ間の移動をともなうリネームを行う場合では，ファイル名のリネームの他にキーワードの付け替えをおこなうようにする．また，同じ仮想ディレクトリ内でファイルでリネームを行った場合では仮想ディレクトリの移動をともなわないので，単にファイル名のリネームのみをおこなう．

4.5.2 ファイルのコピー

同じ仮想ディレクトリ内でファイルのコピーを行った場合，このコピーはコピー元のバックアップ，もしくはファイル自身の複製という意味合いが強くなる．このとき，コピー先のファイルは，コピー元のファイルの複製なのでコピー先ファイルの属性もコピーされるべきである．したがって，同じ仮想ディレクトリでコピーしたときはコピー元がもっているキーワードもコピーするようにする．

一方，同じ仮想ディレクトリ以外でファイルのコピーを行う場合，このコピーは新たに作成したファイルにコピー元ファイルの内容のみのコピーという意味合いが強い．このときのコピーで新たに作られたファイルは，ファイルの複製ではなくコピー元のファイルとは関係のない別物であると判断できるため，ファイルの属性はコピーされるべきではない．したがって，同じ仮想ディレクトリ以外でファイルのコピーをしたときはコピー元がもっているキーワードをコピーしない．

4.6 提案手法の制約

本提案手法は，従来の木構造ファイルシステムでは扱いずらい点を解消するために考案したが，逆にもとから木構造ファイルシステムに適しているものをこの提案手法で扱おうとすると問題が生じる．

この提案手法では，1 つのファイルに対してキーワードの重複は許されないという制限がある．なので木構造ファイルシステムでは実現可能なものも，本提案手法では実現できないことがある．例えば，木構造ファイルシステムでは”/岐阜/岐阜/会議資料.pdf”というファイルを作成することができる．木構造ファイルシステムでは，作成したファイルパス 1 つのみでしかアクセスできないので，同じ文字列であっても階層によって違う意味を持つことができる．この例では，上位の階層の”岐阜”は岐阜県，下位の階層の”岐阜”は岐阜市を意味している．

しかし，本提案手法では，ファイルを指定する際に構成要素であるキーワードを任意の順で指定可能であるため，キーワード出現する順番によって別の意味を与えることはできない．なので本提案手法においてはユーザ自身がキーワードごとに違う意味を

与えなければならない．この例では”/岐阜県/岐阜市/会議資料.pdf”のように違うキーワードを割り当てなければならない．

第5章

実装と評価

5.1 実装概要

4章で提案したファイルシステムを実装するには，カーネル内に実装するかユーザレベルファイルシステムとしての実装が考えられる．現在，多くのユーザレベルファイルシステムが実装されている．その理由として，ユーザレベルファイルシステムは実装の容易さと，十分な速度と機能を有しているという点があげられる．したがって，本研究ではユーザレベルファイルシステムを利用することとした．

本提案手法をユーザレベルファイルシステムとして実装するために，ユーザレベルファイルシステム構築ツールのFUSE[5]とデータベースにMysql[6]を用いた．

5.1.1 FUSE

FUSE(Filesystem in Userspace)とは，ファイルシステムをユーザプログラムとして作成するオープンソースソフトウェアのことである．FUSEは，FUSE自身がファイルシステム機能を持つのではなく，作成したファイルシステムプログラムと通信する機能のみを持つ．FUSEを利用することにより，open，read，writeのようなシステムコールはアップコールの形に変換されユーザプログラムに渡される．このFUSEを用いてファイルシステムを作成するには，FUSEのカーネルモジュールから渡される複数の関数を作成すればよい．

5.1.2 Mysql

Mysql は，オープンソースの RDBMS(リレーショナルデータベース管理・運用システム) の 1 つである．Mysql は世界でもっとも有名なオープンソース・データベースとして知られている．他の RDBMS にくらべ高速に動作するため，特に更新よりも参照頻度が高いアプリケーションに有効であるとされている．さらに，扱いやすいことから非常に多くの Web サイトで使用されている．

また，Mysql は，非常に多くのプラットフォームに対応しており，Linux や FreeBSD などの UNIX 系プラットフォームはもちろんのこと，Windows といった非 UNIX 系プラットフォームにも対応している．

5.2 システムの構成

FUSE と Mysql を用いた提案手法の構成図を図 5.1 に示す．

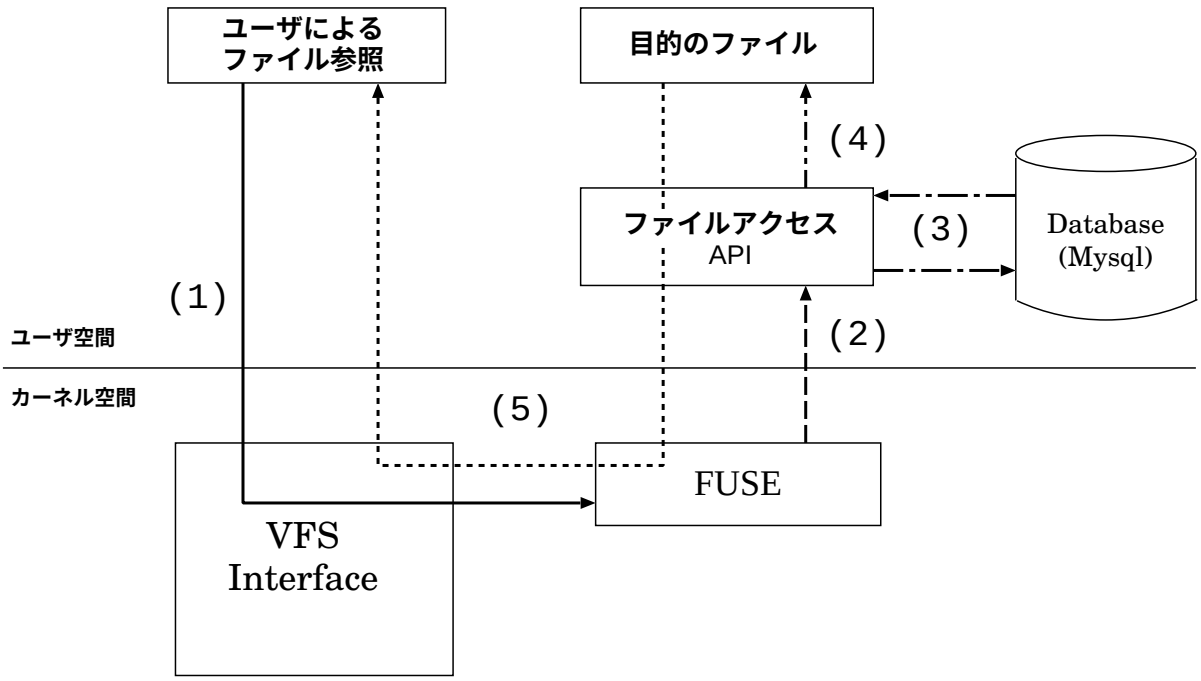

図 5.1: 提案手法のシステム構成

ファイルアクセスの流れを open システムコールを例にとって説明する．また，open

システムコールで開くファイルを，”/検討会資料,11 日/index.html”とする．このとき，ファイルアクセスの流れを，次に示す．

(1) open システムコールは，VFS(Virtual File System) のインターフェースを介し FUSE のカーネルモジュールにアクセスする．

(2) この FUSE のカーネルモジュールはこのシステムコールをアップコールとして提案手法を実装したファイルアクセス API に渡す．

(3) 渡されたシステムコールはデータベースにキーワード”検討会資料”と”11 日”を含んだ”index.html”というファイルがあるか問い合わせを行う．

(4) 問い合わせを行い，結果が 1 つに限定できたときに，そのファイルを開く．

(5) open システムコールの結果をユーザに返す．

5.3 データベース管理とファイル実体

ファイル名とキーワードはデータベースに図 4.1 のように管理されるが，ファイルの実体をこのままファイル名で管理すると同名ファイルが存在する場合，ファイルが作成できない．このため，ファイル実体のファイル名は，キーワードとユーザが付けたファイル名に対するハッシュ値で管理する．

ハッシュ関数にかける際，ファイル名とキーワードは文字コードにより辞書式順序にソートする．こうすることにより，キーワードとファイル名の組合せ方によりハッシュ値が異なることを防ぐことができる．次の 2 通りのファイル作成方法を例に，ソートの必要性を説明する．

%touch 検討会資料,6 月,11 日,index.html

%touch 6 月,11 日, 検討会資料,index.html

これらは，キーワードの並び順に意味はないので，同じファイルを指す．しかし，それぞれの状態でソートせずにハッシュ値に変換すると，ふたつのハッシュ値は異なる

ものとなる．このハッシュ値が異なることにより 2 つのファイルは別のものとして認識されるので，2 つは同じファイルであるということに矛盾を生じてしまう．

また，キーワードの前後を”,”で区切って管理するようにした．このようにすると，複数のキーワードを 1 つの文字列として管理できるようになる．例えば，キーワード”6 月”をもったファイルを検索しようとしたとき，ワイルドカードを用いて次のように問い合わせることで，1 つの文字列内でも指定したキーワードの検索が可能となる．

SELECT from list where キーワード LIKE ’%,6 月,%’

これらのことを適用した実際のデータベースでは図 5.2 のように管理される．

ファイル名	キーワード	ハッシュ値
kurimoto-ken-20070618.ppt	, 検討会資料,6 月,18 日, 栗本,	EDE8FE91E3B4···
kurimoto-paper-20070925.ppt	, 論文紹介,9 月,25 日, 栗本,	E3FD45E81E21···
takagi-ken-20070501.pptx	, 検討会資料,5 月,1 日, 高木,	561868C16129···
index.html	, 検討会資料,6 月,11 日,	B6B215F669AC···
index.html	, 検討会資料,6 月,25 日,	1803FBB4CCFD···

図 5.2: 提案システムにおけるファイル名管理

5.4 ファイルアクセス API

ここでは，自作した主要なファイルアクセス API について説明する．

5.4.1 getattr

getattr 関数はファイルやディレクトリ情報を取得する関数である．

仮想ディレクトリの場合

与えられたパスがキーワード (拡張子無し) だった場合，まずデータベースにそのキーワードを持つファイルがあるか問い合わせを行う．問い合わせをした結果，そのキーワードを持つファイルが存在したなら，それは仮想ディレクトリであると判断し，ディ

レクトリ情報を返す．キーワードを持つファイルが存在しなかった場合そのキーワードは存在しないのでエラーを返す．

ファイルの場合

FUSE によって与えられるパスがファイル (拡張子有り) だった場合，まずデータベースにそのファイルがあるか問い合わせを行う．問い合わせをした結果，そのファイルが存在したなら，そのファイル実体の情報を返す．キーワードを持つファイルが存在しなかった場合そのファイルは存在しないのでエラーを返す．

5.4.2 readdir

readdir 関数は，仮想ディレクトリの中にあるファイルやディレクトリの名前を返す関数である．

ルートディレクトリ内の場合

マウントされたファイルシステムのルートディレクトリ内だった場合，キーワードは何も指定されていないので，全てのファイル名とキーワード名を返す．

ただし，同じファイル名が存在した場合，そのファイル名を返さない．これは，第 4 章で述べたように仮想ディレクトリ内に同名ファイルが存在したときの混乱を防ぐためである．

仮想ディレクトリ内の場合

仮想ディレクトリ内だった場合，その仮想ディレクトリ名のキーワードを含むファイルが存在するかデータベースに問い合わせを行う．問い合わせを行った結果，ファイルが存在したならそのファイル名と仮想ディレクトリに使われたキーワード以外のキーワード名を返す．

またルートディレクトリの時と同じように，問い合わせた結果同じファイル名が存在した場合は，そのファイル名は返さない．

5.4.3 mknod

mknod 関数はファイルを作成する関数である．

まずFUSEから与えられるパスを参照し，そのパス内にあるキーワードを含んだファイルが存在するかどうかをデーターベースに問い合わせる．問い合わせを行った結果，ファイルが存在しなかった場合のみファイル作成を行い，1 つでも存在した場合はエラーを返す．次に，パスからキーワードとファイル名を抽出し，これでファイル実体名のためのハッシュ値を作成する．そして，このファイル名とキーワードとハッシュ値を新たにデータベースに挿入する．

5.4.4 rename

rename 関数はファイルやディレクトリの名前や位置を変更する関数である．

ファイルの場合

ファイルのリネームの場合，まずファイル名のリネームを行う．次に，変更前のファイルパスに書かれているキーワードを取り除き，変更後のファイルパスに書かれているキーワードを付加する．それから，変更したファイル名とキーワードでファイル実体名のためのハッシュ値を作成する．そして，データベース内にある変更前のファイル名のデータを変更した内容に更新する．最後に，ファイル実体のリネームを行う．

仮想ディレクトリの場合

仮想ディレクトリのリネームの場合，仮想ディレクトリ名のキーワードを含んだファイル全てに対してそのキーワードのリネームをおこなう．

まずデータベースにその仮想ディレクトリ名のキーワードを含んだファイルをすべて抽出する．次に，抽出したファイルすべてに対して，変更前のパスに書かれているキーワードを取り除き，変更後のパスに書かれているキーワードを付加する．それから，変更したファイル名とキーワードでファイル実体名のためのハッシュ値をそれぞれ作成する．そして，データベース内にある変更前のファイル名のデータを変更した

内容に更新する．最後に，ファイル実体のリネームをそれぞれ行う．

5.4.5　unlink

unlink 関数はファイルを削除する関数である．

まずFUSEから与えられるパスを参照し，そのパス内にあるキーワードを含んだファイルが存在するかどうかをデーターベースに問い合わせを行う．問い合わせを行った結果，ファイルが 1 つに特定できた場合，データベース内のファイル情報とファイル実体を削除する．また，問い合わせ結果が 2 つ以上存在したり，無かった場合にはエラーを返す．

5.4.6　open，read，write

open,read,write のそれぞれの関数はファイルのオープン，読み，書きを行う関数である．

まずFUSEから与えられるパスを参照し，そのパス内にあるキーワードを含んだファイルが存在するかどうかをデーターベースに問い合わせを行う．問い合わせを行った結果，ファイルが 1 つに特定できた場合，そのファイル実体に対して open，read，write システムコールを呼び出す．また，問い合わせ結果が 2 つ以上存在したり，無かった場合にはエラーを返す．

5.5　キーワードの継承

本提案システムでは，ファイルの作成などはファイルパスをみてキーワードとファイル名の登録を行っている．このため，必要なキーワードが継承されないという可能性がでてきてしまう．例えば，エディタなどの一部のアプリケーションではバックアップとして元のファイルをリネームし，元のファイル名で新たにファイルを作成するものが存在する．この場合，新しいファイルはファイルパスをみて作成するので元ファイルに存在するキーワード全てを付加できるとは限らない．また，4.5.2 項に述べたように，キーワードの継承を必要とするファイルのコピーを行った場合でも，コピー先

ファイルを作成する際にファイルパスを参照してしまうので，ファイルパスに付いているキーワードしか付加されない．したがって，これらの場合について元ファイルが持っている全てのキーワードを継承させる必要がある．

しかし，ファイルシステムのみでキーワードの継承を行うのは容易ではない．なぜなら，ファイルシステム側では，単にシステムコールが呼ばれるだけなので，ファイルをコピーやバックアップをとっているかどうかは判断できないからである．例えば，ファイルのコピーを行った場合，ユーザは単にファイルをコピーしてる様にしか見えない．ところが，ファイルシステム側では，コピー元を read システムコールにより内容を読み取り，その内容をコピー先に write システムコールで書き込むという処理を行っている．つまり，ファイルシステム側ではファイルのコピーをおこなっても read, write といったシステムコールしか呼ばれない．そのため，ファイルシステム側ではコピーとそれ以外の処理を区別することができない．

このような問題を解決するために機能の改良を行った．エディタなどのファイルのバックアップを行うアプリケーションの場合については，同一プロセスがファイルをリネームした後にファイルを作成した場合にのみキーワードの継承を行うようにした．方法として rename 関数が呼ばれたときにそれを呼び出したプロセス ID とファイル名とキーワードを記憶する．そして，mknod 関数が呼ばれたときに，呼び出したプロセス ID とファイル名で検索しそれが存在した場合にキーワードの継承を行う．

しかし，コピーの場合では，4.5.2 項に述べたようにバックアップの時と違い，必ずキーワードを継承するという訳ではない．しかも，ファイルシステム側では read，write といったシステムコールしか呼ばれないので，キーワードの継承を必要とするのかどうかはわからない．したがって，ファイルコピーの際，任意にキーワードの継承をすることは本システムでは不可能である．よって本システムでは，別途キーワードをコピーするアプリケーションを作成した．キーワードの継承を必要とするコピーを行う場合，シェルのエイリアス機能を用い，cp コマンドの後にそのアプリケーションを連動させるようにしてもらうしかない．

5.6 動作

前節までに述べてきたことを実装し，そのシステムの動作の確認をおこなった．複数のファイルを管理し実際にファイルのブラウジングを行う．

また，今回の動作実験では，既存アプリケーションのプログラムを変更することなく使用することができることを確認するため，samba を用いて Windows マシンから実装したファイルシステムにアクセスすることにした．

5.6.1 動作モデル

今回使用す本研究室内の本年度の発表資料 248 個を用いた．また，ファイルに以下のキーワードを付加した．

- 作成者
- 作成者の学年
- 発表資料の種類 (研究内資料，各論文発表会など)
- 発表年，月，日

5.6.2 動作環境

評価に使用するサーバ側とクライアント側の環境を表 5.1 に示す．また，本提案システムで利用する FUSE は Ver2.7.0，Mysql は Ver5.0.27-0vl3 をそれぞれ使用した．

表 5.1: サーバ側とクライアント側の環境

	サーバ側	クライアント側
OS	Vine Linux 4.2	Windows XP Professional SP2
Kernel	2.6.16-0vl76.28	
CPU	Pentium4 3.40GHz	PentiumM 1.00GHz
メモリ	2GB	768MB

5.6.3 動作結果と考察

本提案システムマウントした時のルートディレクトリを図 5.3 に示す．図 5.3 を見ると，”￥test”というのがこの提案システムにおけるルートディレクトリを示している．ルートディレクトリ内には全てのファイルとファイルが持っているキーワードが仮想ディレクトリとして表示されているのがわかる．

次に，仮想ディレクトリ”/栗本/検討会資料”の内容を図 5.4 に示す．図 5.4 を見ると，”/栗本/検討会資料”内にはキーワード”栗本”と”検討会資料”が含まれているファイルとそのファイルが持っているキーワードが表示されているのがわかる．このとき，”kurimoto-ken-20070618.ppt”にアクセスするという場合，ファイルパスは”/栗本/検討会資料/kurimoto-ken-20070618.ppt”になり全てのキーワードを指定しなくてもファイルにアクセスすることが可能となる．

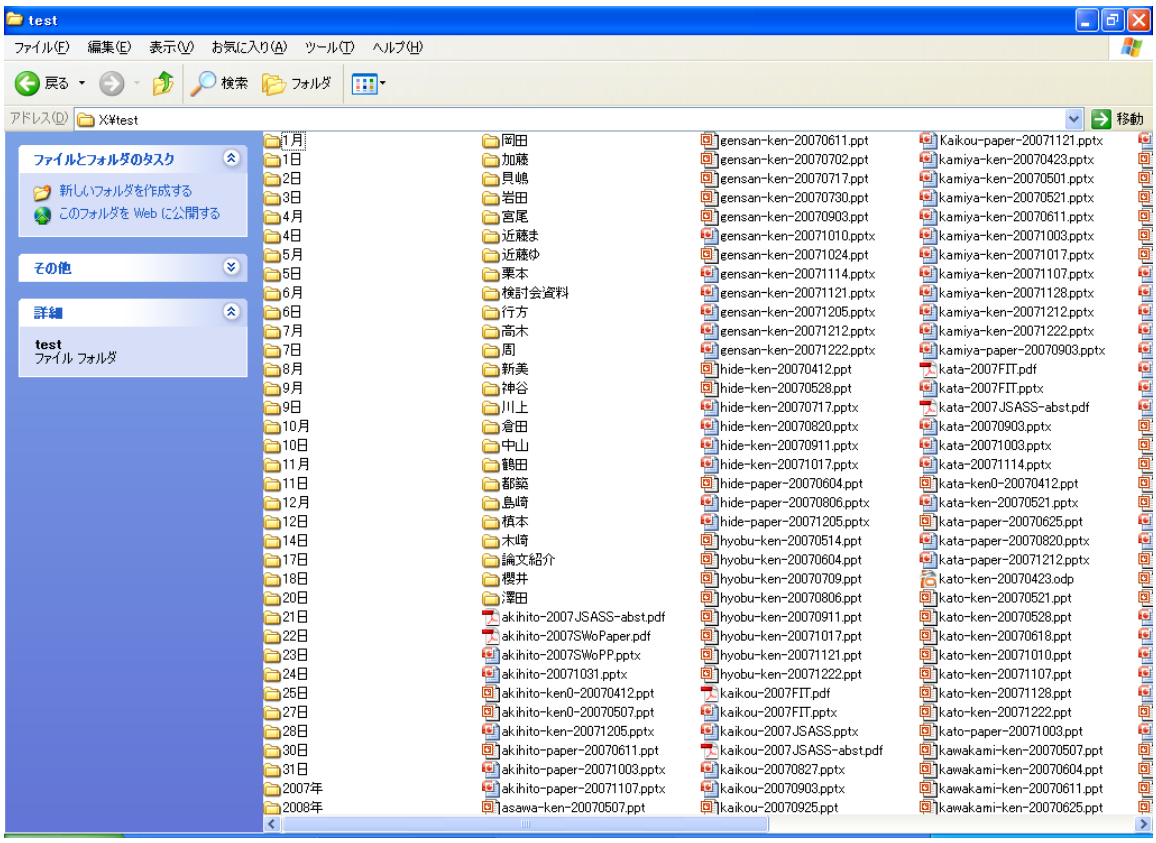

図 5.3: ルートディレクトリの内容

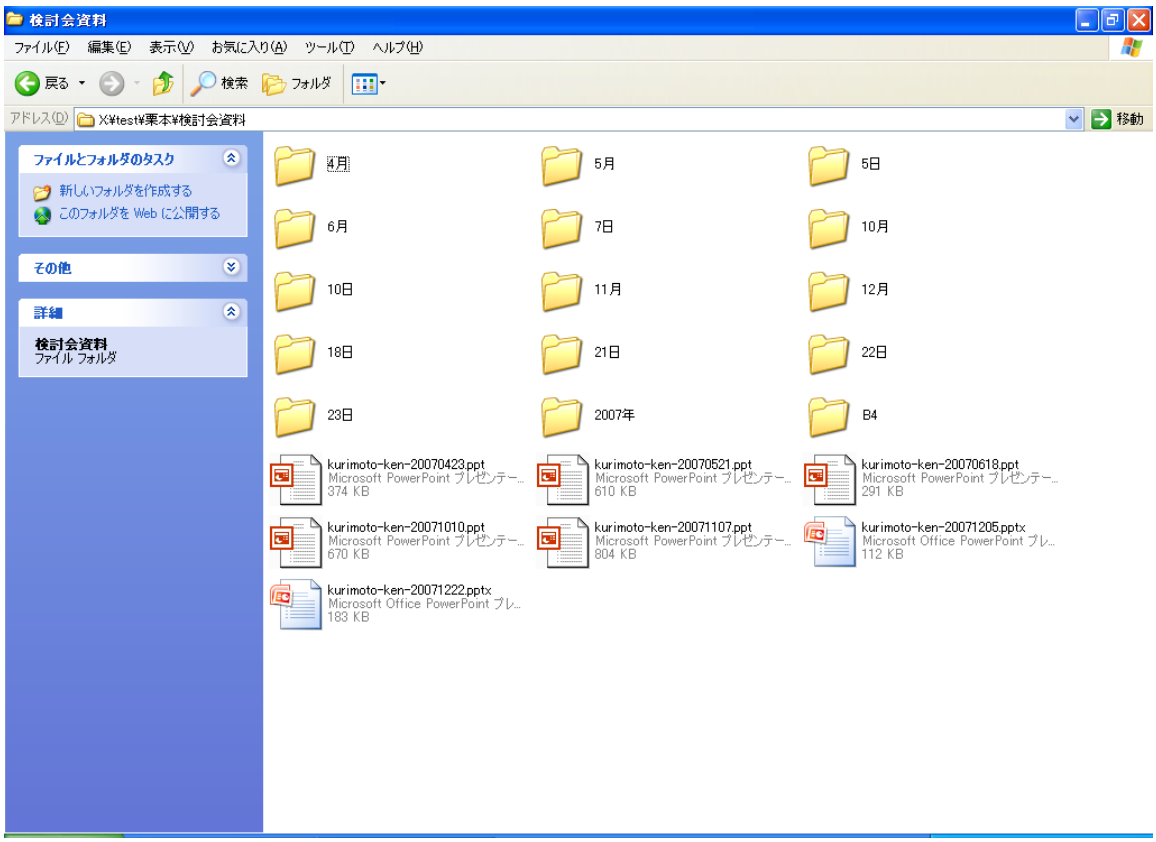

図 5.4: ”/栗本/検討会資料”の内容

次に，仮想ディレクトリ”/検討会資料/栗本”の内容を図 5.5 に示す．図 5.5 を見ると，”/検討会資料/栗本”内にはキーワード”検討会資料”と”栗本”が含まれているファイルとそのファイルが持っているキーワードが表示されるが，この内容は図 5.4 の時と同じである．この図より，本提案システムではキーワードの順序を入れ換えることが可能であるのがわかる．

また，これらの動作図を見ると，仮想ディレクトリの数が多く，見づらくなっている．これは，検索結果にでてくるファイルのすべてのキーワードを仮想ディレクトリとして表示されるからである．

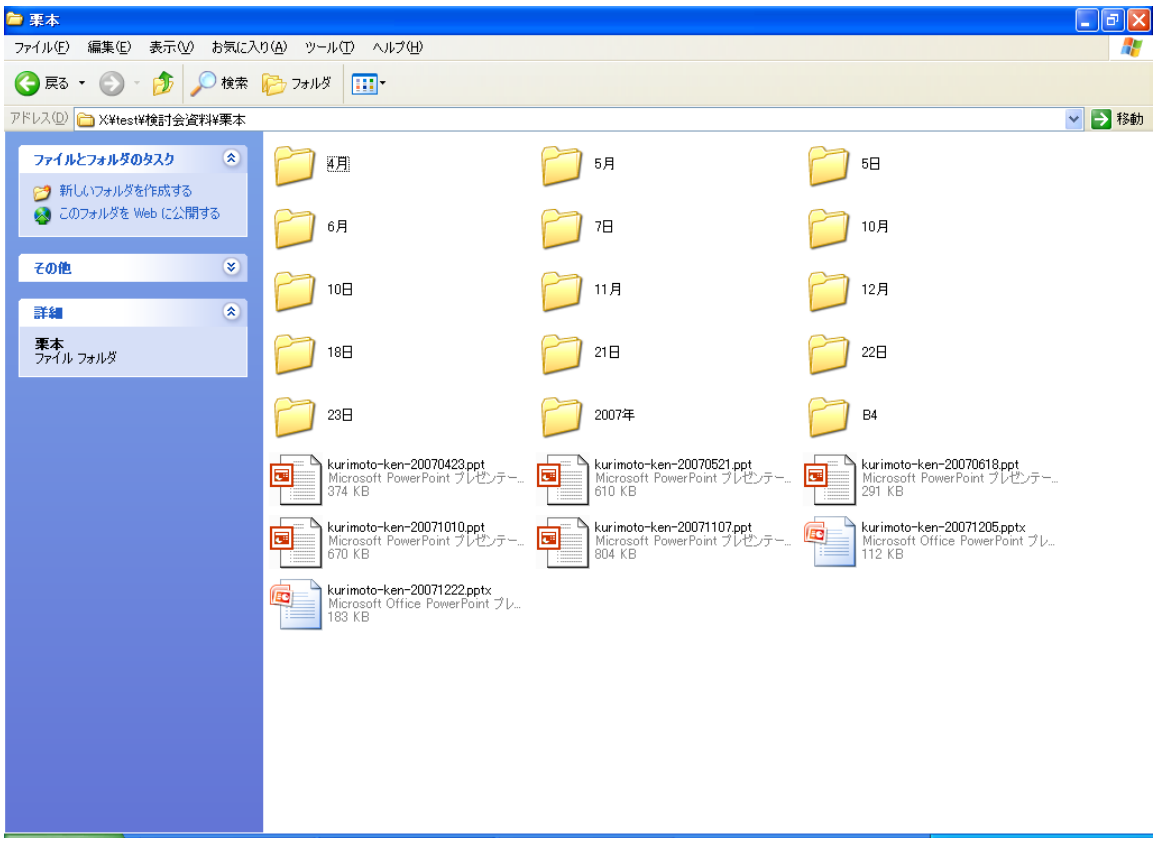

図 5.5: ”/検討会資料/栗本”の内容

5.7 評価

本提案手法を適用したときのオーバーヘッドの計測を行う.

5.7.1 評価方法

オーバーヘッドを計測するために，ディレクトリ参照に要する時間を計測した．今回、評価に用いるディレクトリ参照の手順は次のとおりである.

1. opendir() 関数を用いてディレクトリを開く
2. ディレクトリの内容を NULL になるまで readdir() 関数を用いて取得する

提案手法のオーバーヘッドを測定するために，次の三種類についてそれぞれディレクトリ参照を 1000 回試行し，平均時間を計測する．

方法 1 Linux の標準的なファイルシステム

方法 2 Linux の標準的なファイルシステムを FUSE でマウントしたもの

方法 3 提案手法を FUSE でマウントしたもの

5.7.2 評価環境

評価に使用するマシンは表 5.1 に示したサーバ側のマシンを用いた．ディレクトリ参照に用いるディレクトリには，ファイルが 248 個，ディレクトリが 70 個格納されているものを用意した．

5.7.3 結果と考察

測定結果を表 5.2 に示す．

表 5.2: ディレクトリ参照に要する時間

	方法 1	方法 2	方法 3
参照時間 [msec]	4.257	5.896	7.013

方法 1 と方法 2 の参照時間を比較すると，方法 2 は方法 1 の約 1.4 倍の時間がかかっていることがわかる．したがって，FUSE を使用すると約 1.4 倍の時間がかかるということがわかった．

次に，方法 2 と方法 3 の参照時間を比較すると，方法 3 は方法 2 の約 1.2 倍の時間がかかっていることがわかる．これにより，本提案手法が Linux の標準的なファイルシステムにくらべ，本提案手法は約 1.2 倍の時間で済むというのがわかった．この参照時間の差は本提案手法がデータベースに問い合わせをする時間や同名ファイル名の解決や，キーワードを成形するための時間だと思われる．

第6章

まとめ

本研究では，従来の木構造に従ったファイルシステムではなく，キーワードベースによる検索機能をもったファイルシステムを提案した．また，この提案システムをユーザレベルファイル構築ツールである FUSE とリレーショナルデータベース管理システムの Mysql を用いて実装した．そして，実装した提案システムに実際にファイルを登録し，ディレクトリ参照のオーバーヘッドを計測した．評価した結果，本提案手法は，Linux の標準的なファイルシステムと比べ，約 1.2 倍の増加で済むことができた．

今後の課題として，表示される仮想ディレクトリの数の抑制が挙げられる．5.6.3 項に述べたように，検索結果に現れてくるファイルのすべてのキーワードを仮想ディレクトリとして表示されるため，仮想ディレクトリの数が多くなり見づらくなってしまう．これを抑制するためには，キーワードに順序関係をもたせるという方法が考えられる．例えば，ファイルのキーワードに月と日にちが付けられているとき，日にちのキーワードは月の仮想ディレクトリ内でしか表示されないようにする．このようにすることで，順序の高いキーワードしか表示されなくなり，表示される仮想ディレクトリ数を制限することが可能となる．

また，本提案手法ではファイルにキーワードを付けるにはユーザが手動で行わなければならなず，ユーザ側からしてみれば大変な手間となってしまう．したがって，ファイル内容や作成日時などをみて自動的にキーワードを付加するようにしていきたい．

謝辞

本研究のために多大な御尽力を頂き，日頃から熱心な御指導を賜った名古屋工業大学の齋藤彰一准教授に深く感謝致します.

また, 本研究の際に多くの助言，協力をして頂いた松尾啓志教授，津邑公暁准教授，松井俊浩助手，及び齋藤研究室ならびに松尾・津邑研究室の皆様に深く感謝致します.

参考文献

[1] 滝田 裕, 多田 好克 ”全文検索エンジンを利用したファイルシステムの名前空間拡張”,
情報処理学会論文誌 Vol47 No.SIG 3(ACS 13) 2006.3.

[2] 新城 靖, 西尾 克己, 板野 肯三 ”SetNS:記号集合に基づく名前サービス”,
情報処理学会論文誌 Vol44 No.SIG 11(ACS 3) 2003.8.

[3] Microsoft Corporation:Windows Desktop Search,
http://www.microsoft.com/windows/products/winfamily/desktopsearch/default.mspx.

[4] Google:Google Desktop,
http://desktop.google.com/.

[5] SourceForge project:FUSE,
http://fuse.sourceforge.net/.

[6] MySQL AB:Mysql,
http://www.mysql.com/.