SetNS：記号の集合に基づく名前サービス

新城 靖[†] 西尾克己[††] 板野肯三[†]

この論文は，名前サービス SetNS（Set Name Service）の概念とファイル・システムにおける利用について述べている．SetNS では，名前は，記号（要素文字列）の集合として指定される．利用者は，名前を指定するとき，それを構成する記号を任意の順序で与えることができる．さらに利用者は，長い名前を省略形で指定することもできる．SetNS を Unix のファイル・システムにおいて通常のコマンドにより利用可能にするために，SetNS は，仮想ディレクトリ，コンテキスト，完全名，および，部分名という概念を導入している．例題として World Wide Web 用のディレクトリ・サービスのデータがそのファイル・システム上に再構築されている．再構成されたディレクトリ・データにより，二次記憶の使用量や入出力回数を評価している．

SetNS: A Name Service Based on Set of Symbols

Yasushi Shinjo,[†] Katsumi Nishio[†] and Kozo Itano[†]

This paper describes the concept of SetNS (Set Name Service) and its usage in a file system. In SetNS, a name is specified with a set of symbols (element strings). When a user specifies a name, the user can give symbols constructing the name in arbitrary order. Furthermore, a user can abbreviate a long name. To use SetNS in a file system, SetNS introduces the ideas of contexts, full names, and partial names. As an example, data of a directory service for the World-Wide Web is reconstructed in the file system. The usage of secondary storag and the number of I/O are evaluated by using the constructed directory data.

1. はじめに

コンピュータ・システムにおいて**名前サービス（name service）**は，さまざまな場所で重要な役割を担っている．名前サービスとは，文字列のような高いレベルの名前を，整数のような低いレベルの名前へ変換するサービスである．たとえば，ファイル・システムでは，利用者から与えられた文字列による名前がシステムの内部で使われる番号や構造体へ変換される．インターネットの DNS（Domain Name System）では，文字列で表現されたホスト名が整数である IP アドレスに変換される．

現在広く使われている名前サービスでは，そのモデルとして主にフラットな構造か木構造が使われている．フラットな構造は，単純でわかりやすく，実現も容易である．しかしながら，人間がフラットな構造として把握することができる要素数には限界がある．木構造は，フラットな構造と比較して高い表現能力を持っており，住所や組織など，もともと木構造に基づいている情報を扱うときには特に有効である．要素数が増えたときにも，効率良く名前サービスを実現する方法が知られている．

木構造に基づく名前サービスでは，名前を付けるときに 1 通りの視点しか与えられないという問題がある．このため，もともと独立した概念を無理やり木構造に押し込める必要がある．たとえば，「コンピュータ（`Computers`）」と「教育（`Education`）」は，独立した概念であるが，木構造で表現するときには，`"/Computers/Education"`，または，`"/Education/Computers"`のいずれかの順序を選択しなければならない．この階層の順序が自明ではないときには，全体で首尾一貫して名前を付けることや登録されているものを探すことが難しくなる．さらに，純粋な木構造では，利用する局面に応じて階層の順序を動的に入れ替えることはできない．

木構造に基づく名前サービスのもう 1 つの問題点は，細かく分類すると階層が深くなりすぎることである．たとえば，Qwerty 式のキーボードのために，次

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 筑波大学大学院博士課程工学研究科
Doctoral Program in Engineering, University of Tsukuba

のような長い名前が付けられることがある．

```
"/Computers/Hardware/Peripherals/\
 Keyboards/Qwerty"
```

このような長い名前は，ある階層，たとえば，"/Computers/Hardware/"以下を検索するときに便利である．一方，これをもっと短い名前，たとえば，"/Computers/Keyboards/Qwerty"でもアクセスしたいという要求がある．

多くのファイル・システムでは，階層の順序の問題と長すぎる名前の問題を緩和するために木構造に部分的に DAG（Directed Acyclic Graph）を取り入れている．たとえば，Unix では，ファイルの名前は，基本的には木構造で管理されるが，シンボリック・リンクの導入により，DAG となる．上で述べた階層の入換えの例では，"/Computers/Education"に対して"/Education/Computers"というシンボリック・リンクを作成すれば，2 つの方法でアクセス可能になる．長すぎる名前の問題を緩和するために，シンボリック・リンクにより短い別名を与えることもしばしば行われる．上で述べた例では，"/Computers/Keyboards/Qwerty"から長い名前へシンボリック・リンクを作成すればよい．

このようなシンボリック・リンクを作成する方法は，階層が浅く入れ替える必要がある項目が少ないうちや，短い名前を与えるべき項目が少ないうちには非常に有効な方法である．しかし，階層が深くなり，入れ替える必要がある項目が増えてきたり，短い名前を持たせるべき項目が増えてくると，あらかじめシンボリック・リンクを作成することは次第に困難になっていく．

このような問題点を解決するために，木構造とは異なるモデルに基づく名前サービス SetNS を提案する[9),10)]．**SetNS（Set Name Service）**では，名前は，**記号の集合（set of symbols）**として与えられる．記号とは，区切り文字以外の文字からなる文字列である．木構造に基づく名前サービスと比較して SetNS は，次のような特徴を持つ．

- 1 つの名前を構成している記号を，任意の順序で指定することができる．
- 長い名前を省略形で指定することができる．

逆に，SetNS には次のような制限がある．

- 1 つの名前の中で記号の重複は許されない（異なる名前で共通の記号を用いることは許される）．

この論文では，名前サービス SetNS の概念とそのファイル・システムにおける利用について述べる．Unix のファイル・システムを操作するための標準的なコマンドにより円滑に SetNS を利用するために，仮想ディレクトリ，コンテキスト，部分名，および，完全名といった概念を導入する．

この論文の構成は，以下のようになっている．2 章では，関連研究について述べる．3 章では，SetNS の概念と Unix のファイル・システムにおける利用について述べる．例題として，World Wide Web（WWW）用のディレクトリ・サービスのデータを用いる．4 章では，索引を用いた SetNS の実現方法の 1 つを示す．5 章では，必要な二次記憶の量と入出力回数という側面から SetNS を評価する．

2. 関 連 研 究

2.1 WWW 用のディレクトリ・サービス

WWW（World Wide Web）用のディレクトリ・サービスでは，利用者は，基本的には木構造に基づくメニューを逐次的に選択していくことで目的のページを探す[6),14)]．このとき，完全な木構造ではなく，HTML のハイパーリンク機能を利用して，複数の視点から目的のページを探せるようにしている．たとえば，The Open Directory Project [6)] により維持されているディレクトリには，次のような節がある．

```
/Computers/Education/Operating_Systems@
(1)
```

この例で，"Operating_Systems"には，下の階層ではなく，次の場所へのハイパーリンクが埋め込まれている．

```
/Computers/Software/Operating_Systems/\
Education/
```

このようなハイパーリンクによる疑似的な階層の作成は，完全ではない．直接 WWW ブラウザに (1) のような URL を与えても目的のページを得ることはできない．また，次のような項目の順序を入れ替えた階層は存在しない．

```
/Education/Computers/Software/\
Operating_Systems
```

この論文では，このような記号の入替えを許す名前サービスについて述べる．

2.2 属性に基づく名前サービス

いくつかの属性に基づく名前サービスでは，属性と属性値の組の順序を入れ替えることを許している．意味ファイル・システム（Semantic File System）では，ファイルの内容からいくつかの属性を自動的に抽出し，それらを用いた名前解決を可能にしている[3)]．意味ファイル・システムでは，次のような形式の名前を利用することができる．

```
/owner:/smith/text:/resume/bio.txt
```

これは，属性 owner の値が smith で属性 text の値が resume である bio.txt というファイルを示している．このように，意味ファイル・システムでは，属性，値，属性，値と指定していく．属性と値の組を，**属性値対（AV-pair，attribute-value pair）**と呼ぶことにする．X.500 のようなディレクトリ・サービスでも，属性値対を並べて検索することができる[5]．X.500 の属性値対の順序の制約を緩める方法も提案されている[4]．

これらの属性に基づく名前サービスを提供するシステムでは，属性値対の順序には，論理的には意味がない．順序に意味がないという性質は，この論文で述べる SetNS でも同じである．ただし，これらのシステムでは，属性と値が明確に区別され，それらを組にして指定する必要がある．SetNS では，そのような制約はなく，任意の記号を任意の順序で組み合わせて名前を構成することができる．

意味ファイル・システムでは，属性に基づく問い合わせを行うために，**仮想ディレクトリ（virtual directory）**という概念を導入している．たとえば，上で示した例では，/owner:/や/owner:/smith/は，仮想ディレクトリである．各仮想ディレクトリの内容は，属性の場合，許される属性値のリスト，そうではない場合，その問合せにヒットしたファイルの，木構造に基づく名前（シンボリック・リンク）である．HAC（Hierarchy And Content）ファイル・システム[2]では，意味ファイル・システムの機能に加えて，利用者が明示的に問合せの結果を編集することを許している．すなわち，仮想ディレクトリからファイルを取り除いたり，新たにシンボリック・リンクを追加したりすることができる．

仮想ディレクトリの考え方は，本論文で述べる SetNS でも踏襲する．SetNS における仮想ディレクトリは，名前に含まれている記号をブラウズするために使うことができる．さらに，HAC ファイル・システムと同様に，書込み可能であり，新たに名前を登録したり，削除したりすることができる．

文献 8) では，階層構造と属性に基づくアクセスを組み合わせる方法が提案されている．このシステムでは，名前は述語の並びとして解釈される．各述語は，グラフ（主に木構造）におけるリンクの名前か，または，ある属性を持っていることを示す．名前は，左から右へ要素ごとに解釈される．ある要素では，いくつかの制限された文字列の集まりの中から任意の順序で文字列を選び指定することができる．名前の要素が属性に関係している部分では，要素の入替えが可能である．SetNS では，名前の要素の順序にはまったく意味がなく，左から右へ解釈するという制約はない．また本論文では，SetNS を索引を用いたスケーラブルに実現する手法を示す．文献 8) では，多数の名前についてのスケーラブルな実現方法は示されていない．

2.3 記号の並びの入替えを許す名前サービス

属性値対ではなく，記号の並びの入替えを許す名前サービスとしては，SetNS[9] に続いて文献 13) においても提案された．文献 13) のシステムでは，SetNS と同様に，名前はキーワードの集合として与えられ，その順序には意味がない．ただし，ファイルを指定するために，一時ラベルと呼ばれる数を使う必要がある．文献 13) で述べられているシステムでは，特定のプログラムで名前をブラウズできるだけであり，通常のコマンドからキーワードの集合による名前を利用することはできない．また，索引などを用いた効率的な実現方式も示されていない．

3. 記号の集合に基づく名前サービス SetNS

SetNS は，現在，Unix のファイル・システムにおいて利用可能になっている[10]．このシステムでは，標準の（手を加えていない）Unix のコマンドにより SetNS の名前を持つファイルが操作可能になっている．たとえば，ディレクトリの内容を表示する ls（list）コマンドや，シェル csh の内部コマンドである cd（change directory）が利用可能である．この章では，その Unix で利用可能なシステムの実行例を示しながら SetNS の概要について述べる．そして名前サービス SetNS における様々な用語の定義を行う．さらに，木構造に基づく名前サービスと SetNS を機能面から比較する．

この章で示す実行結果は，The Open Directory Project[6] により維持されている WWW 用のディレクトリ・サービスのデータ（以下，Open Directory のデータ）を SetNS に登録して作成したものである．全体で約 310,000 個の名前が登録されている．Open Directory のデータについて詳しくは，5.1 節で述べる．

3.1 SetNS の利用例

SetNS において，名前は，記号を区切り文字 ‘,’ で結合したものである．**図 1** に，記号 Computers と記号 Keyboards を両方とも含んでいる名前を示す．Open Directory のデータには，それら 2 つの記号を両方とも含んでいる名前は，図 1 に示した 3 つだけである．

図 2 に，Unix のコマンド ls と cd によりシステムに登録されている SetNS の名前をブラウズしている様子を示す．図 2 の (1) は，ls コマンドにより（仮想）ディレクトリ"Computers,Keyboards"の内容を

- Computers,Hardware,Peripherals,Keyboards,Dvorak,index.html
- Computers,Hardware,Peripherals,Keyboards,Dvorak,Products,index.html
- Computers,Hardware,Peripherals,Keyboards,Qwerty,index.html

図 1 記号 Computers と Keyboards を両方とも含む名前の例
Fig. 1 The names that include both the symbols Computers and Keyboards (an example).

```
% ls -l Computers,Keyboards                                    -- (1)
 drwxr-xr-x 5 yas 1536 Aug 29 20:14 Dvorak
 drwxr-xr-x 5 yas 1536 Aug 29 20:14 Hardware
 drwxr-xr-x 5 yas 1536 Aug 29 20:14 Peripherals
-rw-r--r-- 1 yas   74 Aug 27 19:18 Products
-rw-r--r-- 1 yas   65 Aug 27 19:18 Qwerty
 drwxr-xr-x 5 yas 1536 Aug 29 20:14 index.html
% cd ,Keyboards                                               -- (2)
% ls
 Business        Electronics     Keypads         Products
 Computers       Hardware        Manufacturing   Qwerty
 Dvorak          Industries      Peripherals     index.html
% cd Computers,                                               -- (3)
% ls
 Dvorak          Peripherals     Qwerty
 Hardware        Products        index.html
% ls -ld ,Education,Computers,index.html                      -- (4)
 drwxr-xr-x 5 yas 1536 Aug 29 20:14 ,Education,Computers,index.html
% ls -ld ,Education,Computers,index.html,:full                -- (5)
-rw-r--r-- 1 yas   37 Aug 27 19:19 ,Education,Computers,index.html,:full
% _
```

図 2 Unix のコマンド ls と cd による SetNS の名前のブラウズ
Fig. 2 Browsing names in SetNS with the Unix commands ls and cd.

表示している．そのディレクトリの内容は，図 1 に示した名前を構成しているすべての記号を並べたものである．ただし，ディレクトリ自身を指定するために使われた 2 つの記号 Computers と Keyboards は，取り除かれている．

与えられた記号をすべて含む名前を検索し，複数の名前が見つかった場合，標準ではその結果はディレクトリになる．一方，1 つの名前だけが見つかったときには，ファイルになる．図 2 の (1) では，名前 "Querty"がファイルとして表示されている．これは，Computers，Keyboards，Querty という 3 つの記号をすべて含む名前はただ 1 つしか存在しないことを示している．"Products"についても同様である．その他の記号は，たとえその記号を Computers,Keyboards に加えたとしても，ファイルとして特定することができないため，ディレクトリとして扱われている．この例ではまた，"Computers,Keyboards,Querty"という 3 つの記号だけでも，6 つの記号から構成される名前を持つファイルを指定することができることを示している．すなわち，長い名前を持つファイルを，短い名前でも指定可能であることを示している．

Unix におけるカレント・ワーキング・ディレクトリと相対パス名と類似の概念として，SetNS では,（プロセスごとの）コンテキスト，およびコンテキスト依存の名前という概念を導入した．図 2 の (2)〜(3) に，その利用例を示す．図 2 の (2) で cd コマンドは，シェル（csh）の内部コマンドであり，chdir() システム・コールを発行する．このシステム・コールは，SetNS では,（プロセスごとの）コンテキストを変更するものとして働く．**コンテキスト**は，名前，すなわち，記号の集合であり，たとえ明示的に指定されていなくても名前の操作において自動的に付加されて使われるもの

である．コンテキストが自動的に付加される名前を**コンテキスト依存の名前**と呼ぶ．それに対して，コンテキストが付加されず，それだけで完結している名前を**コンテキスト独立の名前**と呼ぶ．これらの名前は，それぞれ Unix における相対パス名と絶対パス名に対応する．区切り文字‘,’から始まるものをコンテキスト独立の名前，そうではないものをコンテキスト依存の名前とする．

図 2 の (2) では，cd コマンドでシェルのコンテキストを，",Keyboards"に変更している．このとき，引数にはコンテキスト独立の名前が使われている．そして，ls コマンドによりそのディレクトリの内容を表示している．このとき，ls コマンドは，コンテキスト依存の名前"."を暗黙的に指定している．

次に，図 2 の (3) では，cd コマンドに，コンテキスト依存の名前"Computers,"を与えてそのシェル・プロセスのコンテキストを",Keyboards,Computers"に変更している．そして，ls コマンドによりディレクトリの内容を表示している．表示される記号の並びは，(1) で記号を"Computers,Keyboards"の順に指定したときと同じである．この結果から，記号の順序が入替え可能であることもわかる．

SetNS では，検索において引数で与えた記号ではファイルが特定できない場合，ディレクトリが指定されたものとして解釈される．このため，引数とまったく同じ記号だけから構成された名前を持つファイルをアクセスできなくなることがある．たとえば，図 2 の (4) では，引数として Education, Computers, index.html という記号を指定しているが，それら 3 つの記号を含む名前が複数存在するので，ディレクトリとして扱われている．

引数で示した記号だけを含み，その他の記号を含まない名前を指定するためには，特殊な記号:full を用いる．図 2 の (5) では，引数で指定された 3 つの記号 Education,Computers,index.html だけを含み，他にはどんな記号も含んでいない名前が指定されている．そのような名前は 1 つしか存在しないので，その結果はファイルになっている．

3.2 定　　義

名前とは，1 つ以上の要素文字列を，区切り文字を挟むことで結合したものである．**要素（element）**，あるいは，**要素文字列（element string）**とは，区切り文字以外の文字の並びである．**名前サービス**とは，名前とオブジェクト識別子の束縛を管理し，次のような名前の操作を提供するサービスである．

登録： 名前とオブジェクト識別子の組を保存する．

削除： 登録されている名前とオブジェクト識別子の組を取り去る．

検索（解決）： 与えられた名前からそれと対応しているオブジェクト識別子を求める．

一覧： 条件に適合する名前，または，要素文字列のリストを返す．

ここで**オブジェクト識別子**とは，名前サービスでは解釈されないビット列である．たとえば，ファイル・システムでは inode 番号，インターネットの DNS では，IP アドレスがオブジェクト識別子である．

SetNS は名前サービスの一種である．この論文では，SetNS の名前の区切り文字として‘,’を用いる．SetNS では，個々の要素文字列のことを，**記号（symbol）**とも呼ぶ．SetNS における名前は，記号を区切り文字‘,’を挟み結合したものである．ただし，1 つの名前の中で記号の順序に意味はなく，記号の重複は許されない．このような性質を強調する場合，SetNS における名前を，**記号の集合**と呼ぶ．

SetNS において，登録されている記号をブラウズしやすくするために，**仮想ディレクトリ（virtual directory）**の考え方を導入する．仮想ディレクトリの考え方は，2.2 節で述べた意味ファイル・システムから踏襲したものである．ただし，SetNS ではすべてのディレクトリが仮想ディレクトリであるため，単にディレクトリと呼ぶ．

SetNS におけるディレクトリは，次の 3 つの役割を果たす．

(1) 記号の集合（コンテキスト）を保持する．
(2) (1) に含まれている記号をすべて含む名前のリストを保持する．
(3) 一覧操作の結果として，(2) に含まれているすべての記号を返す．ただし，(1) を除く．

SetNS では，長い名前を簡単に指定できるようにするためにコンテキストという概念を導入する．**コンテキスト**は，名前，すなわち，記号の集合であり，たとえ明示的に指定されていなくても名前の操作（登録，削除，検索，一覧）において自動的に付加されて使われることがあるものである．そのように，コンテキストが自動的に付加される名前を**コンテキスト依存**の名前と呼ぶ．コンテキストが付加されず，それだけで完結している名前を**コンテキスト独立**の名前と呼ぶ．

名前の操作の引数に表れる名前は，完全名と部分名に分類される．**完全名**とは，コンテキストを含めて，名前を構成するすべての記号が指定されたものである．完全名は，名前解決において，たかだか 1 つの名前にマッチする．すなわち，オブジェクトが登録され

ていればその識別子が得られ，登録されていなければエラーになる．**部分名**は，コンテキストを含めて，名前を構成する記号の一部が指定されたものである．部分名は，名前解決において複数の名前にマッチすることがある．

明示的に完全名と部分名を指定するには，名前の中にそれぞれ`:full`と`:partial`という特殊な記号を含める．明示的に指定されていない場合，名前の操作によりどちらかが指定されたものとして扱う．たとえば登録では，必ず完全名が必要である．検索と一覧では，標準では部分名が指定されたものとして扱う．削除でも部分名を指定可能にすると便利である．しかし，操作の誤りによる影響が広範囲に及ぶので，部分名による削除を禁止することも考えられる．SetNSでは，部分名から完全名を求める操作がしばしば必要になる．この操作を**完全化**と呼ぶ．

3.3 木構造に基づく名前サービスとの比較

この節では，SetNSと木構造に基づく名前サービスを機能面から比較し，SetNSを評価する．性能面の評価は，5章で行う．

木構造に基づく名前サービスと比較して，SetNSの第1の利点は，名前を指定する際，名前の構成要素である記号を任意の順序で与えることができることである．これは，木構造に基づく名前サービスでは不可能であったことである．

SetNSの第2の利点は，長い名前を省略形で指定することができることである．これは，部分名という考え方を導入したことによる．これにより，たとえ論理的に付加すべき記号が多くなってしまったとしても，短い名前でアクセスすることが可能になる．

SetNSの第3の利点として，もともとの木構造で様々な場所に散在していた項目であっても，SetNSを使うと簡単に集約することができることがあげられる．たとえば，元のOpen Directoryのデータでは，`"/Education"`という，根の直下の項目はなく，`Education`という項目は，6800以上の場所に散在している．よって，`Education`を含む項目をすべて探すには木全体を検索する必要がある．一方，SetNSでは，`",Education"`という名前で，簡単に`Education`を含む項目を集めることができる．さらに，`",Computers,Education"`のように，(根以外の)任意の位置で集約することもできる．

SetNSの特徴を生かすプログラムは，まず与えられた名前でアクセスし，それがファイルであるかディレクトリであるかを調べ，その結果に応じて柔軟に動作を変える．たとえば，Unixのmanコマンドは，`chmod`のようにコマンドとシステム・コールで同一の名前の項目がある場合，標準ではコマンドのマニュアルだけを表示する．SetNSに対応したmanコマンドでは，1つの項目しかない場合は，それを表示し，複数の項目がある場合にはディレクトリとしてアクセスして一覧表を表示する[10]．このようなプログラムを木構造に基づく名前サービスで実現するには，部分木を検索するか，独自に索引を作成して一貫性を維持する必要がある．また，SetNSでは，たとえば，システム・コールには`api,syscall`，ライブラリには，`api,lib`といった記号を与えることができる．これを使えば，プログラミングを行っている場合，環境変数などで`api`というコンテキストを指定しておくことで，標準でシステム・コールのマニュアルを表示させることもできる．

SetNSの利点を用いず，伝統的な手法でプログラムを作成する場合には，特殊記号`full:`を用いる．伝統的なプログラムでは，`open()`システム・コールを実行する段階でファイルとしてアクセスことがことが決まっている場合が多い．このような場合，`open()`システム・コールの引数には，最初から特殊記号`:full`を含めるようにする．たとえば，`",Education,Computers,index.html"`という名前のファイルを開き，その内容を読み込むプログラムでは，次のように指定する．

```
fd = open(",Education,Computers,index.html,\
:full",O_RDONLY);
read(fd,buf,size);
```

1章で述べたように，木構造に基づく名前サービスには適していない名前を無理に木構造に押し込めようとすると，様々な問題が生じる．SetNSは，そのような問題を解決するために考案したものである．逆に，もともと木構造に基づく名前サービスに適していた名前をSetNSで扱おうとすると，逆の意味で問題が生じる．

SetNSには，1つの名前の中で記号の重複は許されないという制限がある．したがって，木構造で表現可能な名前であっても，SetNSでは表現できないことがある．木構造では，たとえば，`"hiroshima.hiroshima.jp"`のように，同じ文字列に対して階層ごとに別の意味を持たせることができる．この例では，`"hiroshima"`は，上位の階層では広島県，下位の階層では広島市を意味する．SetNSでは，同じ記号を出現位置により別の意味を与えることはできない．別の意味を与えるためには，`"hiroshima-city.hiroshima-pref.jp"`のように，別の記号を割り当てる必要がある．

散在している名前を集約できることは，逆に，一覧

操作で記号をブラウズする際に，1つのディレクトリに含まれる記号の数が多くなってしまうという問題がある．たとえば，ディレクトリ",Education"は，約9,200個の記号を含む．この問題を解決する方法としては，シェルが持っている‘*’や‘?’によるファイル名の置換え機能を積極的に利用する方法がある．SetNSにおいても，その機能はそのまま利用可能である．そのほかに，記号をブラウズするための，SetNS専用のコマンドを作成する方法が考えられる．たとえば，多すぎる記号を表示しない機能や，利用頻度が高い記号だけを表示する機能を持たせることが考えられる．

3.4 改　　名

改名（rename）は，基本的には登録と削除を行うことで実現される．Unixのmv（move）コマンドは，システム・コール rename() を実行するためのシェル・レベルのコマンドである．

3.1節で述べたシステムにおて rename() システム・コールが発行された場合，元の名前を示す引数が完全名ならば，単に登録と削除を一度に行えばよい．しかしながら，部分名が与えられた場合には，1つのシステム・コールで複数のファイルの名前を扱うことは危険であるので，エラーを返し禁止している．これは，unlink() システム・コール（rm（remove）コマンド）において部分名を扱えないように禁止していることに合わせたものである．したがって，Unix標準のmvコマンドで部分名を含んだ名前を扱うことはできない．

そこで，SetNSでは独自のmvコマンドを用意している．このコマンドは，まず完全化を行い，元の部分名にマッチする完全名のリストを得る．次に，それらのリストに含まれている完全名1つ1つに対して rename() システム・コールを発行する．

3.5 WWW ディレクトリ・サービス以外のSetNSが有効な例

3.1節では，SetNSをWWWディレクトリ・サービスに利用する例について述べた．SetNSは，そのほかに，マニュアルの整理に利用することができる[10]．たとえば，Solarisでは，socket() は，ライブラリ関数であり，そのマニュアルは，次のファイルに保存されている．

```
man/sman3socket/socket.3socket
```

SetNSでは，これを次のような名前のファイルに保存する．

```
,api,lib,man,network,sgml_format,socket
```

これらの記号の間には，api/lib を除いて木構造としての上下関係は存在しない．

それから，2.2節で述べた属性に基づく名前サービスにおける名前を，SetNSの名前に対応させて利用することもできる．たとえば，次の名前を考える．

```
/owner:/smith/text:/resume/bio.txt
```

これらの属性値対における属性名と属性値の間に文字‘-’を含めることで，次のようにSetNSの名前に対応させることができる．

```
,owner-smith,text-resume,bio.txt
```

なお，人名 smith のように，他の名前とぶつかりにくいような値の場合，属性名のプレフィックス owner- を外すことも考えられる．

4. 索引とキャッシングを用いたSetNS実現

3章で実行例を示したように，記号の集合に基づく名前サービスSetNSをUnixのファイル・システムにおいて利用可能にした．そのプログラムは，C++言語で記述されており，行数は，約8,000行である．この章では，そのプログラムの内部で用いられている重要なデータ構造とアルゴリズムについて述べる．

単に記号の順序の入替えを許すだけならば，与えられた名前の要素文字列をソートすることで正規化し，それを木構造に基づく名前として保存すればよい．しかしこの方法では，一覧操作や部分名による検索を提供するために，木構造全体を調べる必要があるので，それらの操作が非常に重たくなってしまう．この章で述べる実現方式では，一覧操作と部分名による検索を高速に実現するために，二次記憶上のデータに対する索引付けの技術を用いている．この実現方法は，データベースの索引付けで用いられている転置ファイルと呼ばれる手法と，基本的には同じ方法である．転置ファイルと比較して，ここで述べる方法の特徴は，ディレクトリを中心としたオブジェクト指向の概念を用いていること，および，アルゴリズムにおいてキャッシングが中心的な役割を果たしている点にある．

以下では，データ構造とディレクトリ・オブジェクトの概要，および，ディレクトリ・オブジェクトが持つ手続きのうち，検索，登録，一覧の概要について述べる．

4.1 名前番号表と記号索引

SetNSは，名前番号表，および，記号索引と呼ばれる2つの二次記憶上の索引付けられた表により実現される．

名前番号表（The name number map，NNM）のエントリは，次のようになっている．

(1) 名前番号　登録されている各名前に唯一になるように割り当てられた整数．

(2) オブジェクト識別子　オブジェクト（ファイル）の

アクセスに用いられるビット列．名前サービスでは解釈されない．

(3) 名前 正規化された記号の集合．

この表のエントリは，名前番号をキーとして高速にアクセスできるようにする．

記号索引（The symbol index，SI）の各エントリは，次のようになっている．

(1) 記号 区切り文字を含まない文字列．

(2) 名前番号リスト（nnl，Name Number List） その記号を含む名前の名前番号の並び．最終更新時刻を含む．

この表のエントリは，記号をキーとして高速にアクセスできるようにする．

図 3 に，3 つの名前（図 1 に示したものと同じ）が登録された名前番号表と記号索引の様子を示す．たとえば名前番号 1 の名前は，オブジェクト識別子 101 を持っている．名前に含まれている記号は，文字コード順にソートされ正規化されている．記号索引には，名前として使われているすべての記号が登録されている．各記号は，それを含む名前の名前番号のリストを持っている．このリストは，共通部分を求める操作の高速化のために名前番号の順にソートされている．

今回，索引付けには，ファイル・システムで一般的によく利用される B+木を用いた．B+木の実現には Berkeley DB[1] を用いた☆．Berkeley DB を使ってデータを保存するためには，ポインタを含む構造体をメモリ中の連続した番地に整列化（marshaling）しなければならない．このために，今回は，SunRPC で用いられている XDR[12] を用いた．

4.2 ディレクトリ・オブジェクト

SetNS を提供するプログラムの内部では，3.2 節で述べた操作（登録，削除，検索，一覧）は，**ディレクトリ・オブジェクト**と呼ばれるオブジェクトに対する手続きとして実現されている．ディレクトリ・オブジェクトは，メモリ中にのみ存在する揮発的なオブジェクトであり，二次記憶上のデータである名前番号表と記号索引から完全に再生可能である．

ディレクトリ・オブジェクトは，次のような手続きを持っている．

(1) `register()` 名前とオブジェクト識別子の束縛を作成する．

(2) `lookup()` 名前を検索する．検索の結果，ファイル・オブジェクトの識別子，または，ディレクトリ・オブジェクトの識別子を返す．

(3) `delete()` 名前とオブジェクト識別子の束縛を削除する．

(4) `list()` ディレクトリの内容として記号の並びを返す．

The name number map (NNM)

No.	OID	SymbolSet Name
1	101	{ Computers,Dvorak,Hardware,Keyboards, Peripherals,Products,index.html}
2	102	{ Computers,Dvorak,Hardware,Keyboards, Peripherals,index.html}
3	103	{ Computers,Hardware,Keyboards, Qwerty,Peripherals,index.html}

The symbol index (SI)

Symbol	Name Number List (nnl)
Computers	1,2,3
Dvorak	1,2
Hardware	1,2,3
Keyboards	1,2,3
Peripherals	1,2,3
Products	1
Qwerty	3
index.html	1,2,3

図 3 名前番号表と記号索引
Fig. 3 The name number map and the symbol index.

個々のディレクトリ・オブジェクトは，次のようなデータを持つ．

(1) コンテキスト（context） どのような検索が行われた結果そのディレクトリが作成されたかを示す記号の集合．

(2) 名前番号リスト そのディレクトリが含んでいる名前の名前番号の並び．記号索引の 2 番目のフィールドと同じ型．

名前番号リストには，最後に更新された時刻が含まれており，キャッシュの一貫性のチェックに使われる．

ディレクトリは，次のようなライフサイクルを持っている．

(1) ディレクトリは，手続き `lookup()` の実行において，必要に応じて生成される（詳しくは，4.3.1 項で述べる）．

(2) 名前の追加や削除が行われると，二次記憶上の名前番号表と記号索引が更新される．これにともない，ディレクトリが持っている名前番号リストが古くなることがある．ディレクトリに対する操作を行う前には，必要に応じて名前番号表を作り直す（詳しくは，4.3.4 項で述べる）．

(3) 生成されたディレクトリは，参照されなくなっても即座に破棄するのではなく，キャッシュに保存する．これにより，同一のディレクトリに

☆ 名前番号表には，Berkeley DB のレコード番号（`DB_RECNO`）を用いた．これは，内部的には，B+木が使われている．記号索引には，B+木（`DB_BTREE`）を用いた．2 つの表とも，エントリは，可変長である．

対するアクセスを高速化することができる．

(4) 参照されていないディレクトリは，しばらくアクセスされなかったりメモリが不足してきた場合，キャッシュから取り除き破棄する．このときディスクへ出力すべきデータはない．

このようなライフサイクルの中で，最もコストがかかる操作は，入出力をともなう (1) における生成と (2) における名前番号表の作り直しである．入出力回数については，5 章で述べる．

4.3 ディレクトリ・オブジェクトの手続き

ディレクトリ・オブジェクトが持つ手続きのうち，検索，登録，一覧の概要を示す．削除は，登録と類似している．ただし，データを追加するのではなく，消去する．また，これらの手続きを実現する上で重要な補助手続きである名前番号リストを再構築する手続きについても概要を示す．アルゴリズムの記述には，C++ 言語に似た言語を用いる．

4.3.1 検　　索

検索手続きの概要を以下に示す．

```
Object lookup( SymbolSet name ) {
  if( !nnl_consistent() ) nnl_renew();
  new_nnl = copy_nnl( this->nnl );
  foreach( sym in name ){
    sym_nnl = SI->getnnl( sym );
    new_nnl->and( sym_nnl );
  }
  switch( new_nnl->length() ) {
  case 0:  return NULL ;
  case 1:  return NNM->getoid(new_nnl[0]);
  default:
    full=new SymbolSet(this->context,name);
    dir=new Dir(full,new_nnl);
    return dir ;
  }
}
```

まず自分の名前番号リストが最新かどうかを調べ，古くなっていれば作り直す．名前番号リストが最新かどうかは，最終更新時刻を比較することで判定している．自分自身の名前番号リストをコピーし，新しい名前番号リストの初期値とする．引数で与えられた名前（記号の集合）の各要素について，記号索引を引き，名前番号リストを得る．こうして得られた名前番号リストと作業中の名前番号リストの AND 演算を行う．

こうして得られた名前番号リストの長さが 0，すなわち，AND 演算の結果が空集合なら，そのようなファイルは存在しないので NULL を返す．1 個だけ存在した場合は，ファイルである．その場合，名前番号を使って名前番号表を引き，オブジェクト識別子を調べて返す．複数個ある場合は，新たにディレクトリを作成し☆，それを返す．ディレクトリを作成するには，完全名と名前番号リストが必要である．完全名は，現在のディレクトリが持っているコンテキストと，引数で与えられた記号の集合の和をとることで合成している．

4.3.2 登　　録

登録手続きの概要を以下に示す．

```
Status register( SymbolSet name, Object o ){
  if( this->lookup(name) )  return Error ;
  full = new SymbolSet(this->context,name);
  nn = NNM->register( full,o );
  foreach( sym in full )
     SI->addnn( sym, nn );
  return Ok ;
}
```

与えられた名前で検索を行い，見つかればエラーである．検索と同様に，コンテキストと引数から完全名を得て，これを名前番号表に登録する．結果として，名前番号を得る．完全名を構成する各記号について，記号索引のその記号のエントリに，その名前番号を追加する．

4.3.3 一　　覧

一覧手続きの概要を以下に示す．

```
SymbolSet list() {
  if( !nnl_consistent() ) nnl_renew();
  res = new SymbolSet() ;
  foreach( nn in this->nnl ) {
     ss = NNM->getname( nn );
     res->or( ss );
  }
  res->sub( this->context );
  return res ;
}
```

登録手続きと同様に，ディレクトリが持ってい名前番号リストを最新のものにする．初期値として空集合から始める．自分自身の名前番号リストの各番号について，名前番号表を検索し，名前，すなわち，記号の集合を得る．その集合を，OR 演算により結果に加えていく．OR 演算の結果から現在のディレクトリが持っているコンテキスト（記号の集合）を引き，手続きの結果として返す．

☆ 実際には毎回生成するのではなく，キャッシングを行い，再利用している．

4.3.4 名前番号リストの再構築

名前番号リストを再構築する手続きの概要を以下に示す．

```
nnl_renew() {
  sym = this->context->first();
  this->nnl = SI->getnnl(sym);
  foreach( sym in this->context->rest() ) {
    nnl1 = SI->getnnl( sym );
    this->nnl->and( nnl1 );
  }
}
```

自分自身のコンテキスト（記号の集合）の最初の要素を取り出す．その記号で，記号索引を検索し，名前番号を得，その結果を自分自身の名前番号リストの初期値とする．コンテキストの残りの要素について記号索引を検索し，名前番号を得，自分自身の名前番号リストと AND 演算を行い，絞りこんでいく☆1．

5. 二次記憶量と入出力回数

3.3 節では，木構造に基づく名前サービスと比較しながら SetNS を機能面から評価した．この章では，SetNS の性能を，空間的な側面（二次記憶の量），および時間的な側面（入出力回数）で評価する．

SetNS を実現するために必要な二次記憶の量を調べるために，実験を行った．4.1 節で述べたように，記号索引と名前番号表は，B+木を用いて索引付けを行っている．従って，必要な二次記憶の量は，要素数を n とした時には，$O(n)$ となる．これは，要素数に対して十分なスケーラビリティを持っているといえる．この章では，WWW 用のディレクトリ・サービスのデータという，実用的なデータを SetNS で表現したときに実際にどの程度のデータ量が必要になるかを調べる．そして，SetNS が必要とする二次記憶の量が実用的な範囲に収まっていることを示す．

SetNS の時間的な性能は，二次記憶に対する入出力回数で評価することができる．文献 10) に示したように，Unix のファイル・システムで利用可能な SetNS では，実行時間は入出力により決定される☆2．SetNS の実現では，記号索引のエントリや名前番号表のエントリについてキャッシングを行っている．アクセスの局所性がありキャッシュが有効なときには，入出力は行われず，性能上の問題はない☆3．よってこの章では，キャッシュが無効なときの入出力回数を論じる．

表 1 Open Direcotory から抽出した名前
Table 1 Names extracting from Open Directory.

Number of names	312,000
ASCII representation of names	24.7 M bytes
Number of unique elements	100,553
Average length of an element	12.5 bytes
Average number of elements in a name	8.27
Average length of a name	83 bytes

5.1 The Open Directory Project のディレクトリ・データ

The Open Directory Project[6] が維持している WWW 用ディレクトリ・サービスのデータ（以下，Open Directory のデータ）を SetNS の名前として登録した．元の Open Directory のデータの統計情報を**表 1** にまとめる．

Open Directory のデータは，RDF（Resource Description Framework）[7] により記述されている．その構造を表すデータ☆4のタグ `Topic` から属性 `r:id` を抜き出し，区切り文字を ‘,’ に変え，SetNS の名前とした．ただし，現在の所，要素文字列として ASCII 文字しか扱えないので，ASCII 以外の文字を含むものを取り除いた．また，`"Arts_and_Entertainment"` のように，`"_and_"`により 2 つの単語を含むものは，その 2 つの記号が含まれているものとして扱った．さらに，すべての名前の最初に含まれている要素 `Top` を取り除いた．一方，WWW ページのデータを保持するためによく使われる記号である `index.html` を付け加えた．

表 1 で，名前の数とは，タグ `Topic` の数である．全部で約 31 万個の名前が得られた．名前を単純に ASCII 文字で表現した場合，約 24.7 M バイトになる．要素（elements）とは，3.2 節で定義したように，名前を構成する区切り文字以外の文字の並びである．そのうちユニークなものは，約 10 万個存在した．1 つの要素の長さの平均は，12.5 バイトであった．各名前には，平均で 8.27 個の要素が含まれおり，ASCII 表現では平均 83 バイトになる．

5.2 記号索引と名前番号表の大きさ

5.1 節で述べた Open Directory のデータを，SetNS に登録した．その結果を，表 2，表 3，および図 4 に

☆1 実際には，AND 演算を行う時には，要素数が少ない順に行うなどの工夫を行っている．

☆2 文献 10) の段階では，一覧操作（`ls`）の実行時間に問題があった．その後改良を行い，メモリ中の OR 演算に高速なアルゴリズムを用いるようにしたため，すべての操作において入出力により実行時間が決定されるようになった．

☆3 アクセスに局所性がありキャッシュが有効な場合，CPU の負荷は，木構造に基づく名前サービスよりも SetNS の方が大きい．

☆4 `structure.rdf.u8.gz`

表 2 記号索引の統計情報
Table 2 Statistics of the Symbol Index.

Number of symbols	100,553
Average length of a key	12.5 bytes
B+ Tree page size	8,192 Bytes
Number of B+ Tree levels	3
Total size	23.0 M Bytes
Average entry length	240 Bytes

表 3 名前番号表の統計情報
Table 3 Statistics of the Name Number Map.

Number of names	312,474
length of a key	4 bytes
B+ Tree page size	8,192 Bytes
Number of B+ Tree levels	3
Total size	85.2 M Bytes
Average entry length	286 Bytes

示す．

記号索引は，4.1 節で述べたように，記号をキーとして，その記号を含んでいる名前の名前番号（整数）のリストを引くものであり，B+木として表現されている．**表 2** に示すように，要素数は約 10 万，キーの平均の長さが 12.5 バイトであった．ページサイズが 8,192 バイト☆で，木のレベルは 3 であった．全体の大きさは約 23.0 M バイトであり，平均のエントリの長さ（B 木の中間ノードも含む）は 240 バイトであった．

名前番号表は，4.1 節で述べたように，名前番号（整数）をキーとして，それと 1 対 1 に対応している名前とその名前に関連づけられているオブジェクト識別子を高速にアクセスできるようになっている．索引付けには，Berkeley DB のレコード番号（RECNO）を用いた．これは，内部的には，B+木が使われている．**表 3** に示すように，要素数が約 31 万個，キーの長さが 4 バイトである．ページサイズが 8,192 バイトで，木のレベルは 3 になった．全体の大きさは，約 85.2 M バイトであり，平均のエントリの長さ（B+木の中間ノードも含む）は，286 バイトであった．

図 4 は，名前番号表に登録されている名前の長さ（記号の数）の分布を示している．名前の長さは，7 のものが最も多く，6〜10 で全体の 75%を占める．名前の長さの平均は，8.27 である．

表 2 と表 3 に示したように，記号索引の大きさが 23.0 M バイト，名前番号表が 85.2 M バイト，合計で 108.2 M バイトである．これは，312,474 個の名前を保存するために必要な二次記憶の量である．元の Open

☆ Berkeley DB の標準．

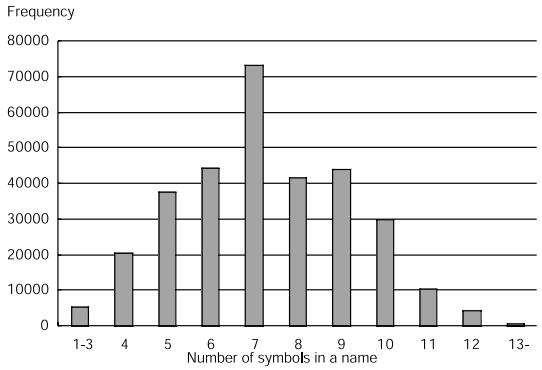

図 4 名前の長さの分布
Fig. 4 The distribution of the length of names.

Directory のデータでは，312,474 個の名前の 1 つ 1 つが，1 つの URL に対応する．その URL で指し示された各 WWW ページは，他のノードへのリンクと，WWW サイトへのリンクを含んでいる．1 つの WWW ページを 8 k バイトと仮定すると，全体の Open Directory のデータの大きさは，約 2,400 M バイトになる．このことから全体のデータの大きさの約 5%で名前を表現できることがわかる．これは，WWW ディレクトリ・サービスという応用では実用上まったく問題がないといえる☆☆．

なお，この実験で用いた Open Directory のデータを Unix のファイル・システムにおいて木構造として表現した場合，約 362,000 個のディレクトリが作られた．ほとんどのディレクトリの大きさは，512 バイトであった．全体で約 220 M バイトの二次記憶の容量を消費した．Unix のファイル・システム（Solaris の UFS）[11] の標準のフォーマットでは，どんなに小さなディレクトリであっても 512 バイトのデータ領域と 128 バイトの inode 領域を消費する．このように，Unix では，2,400 M バイトのデータ量に対して，その約 9%の領域を名前を保存するため当てているが，実用的に使われている．

☆☆ 5%で十分とした根拠としては，まず，単位の曖昧性がある．1 K が 1,000 か 1,024 かで，2,400 MB のデータを格納できる程度の容量のディスクには，5%程度の差が生じるが，現実的にはその程度の差は許容されている．次に，基本ブロック 8 k，フラグメント 1 k の UFS では，1 ファイルあたり平均で 512 バイトの内部フラグメントによる無駄が生じるが，その無駄の合計がファイル数が 31 万個では，約 150 M バイト（2,400 M バイトの 6%）に達する．さらに，UFS では，10%の空きを意図的に設けることで，同一ファイルのブロックを近隣のセクタに配置することを可能にし，高速化を計っている．また，最近のディスクの大容量化と低価格化が進んでいる．大容量のディスクでは，同一型番の製品でも，個体により容量にばらつきがあるので，5%程度のマージンを見込んで利用することは確実性を求める環境ではしばしば行われている．

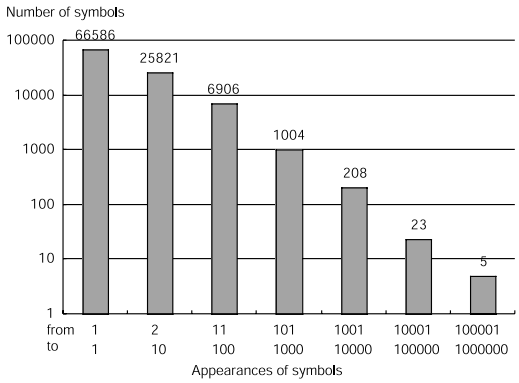

図 5 記号の出現回数の分布
Fig. 5 The distribution of the appearances of symbols.

5.3 入出力回数のオーダ

ここでは，次のパラメータを用いて入出力回数を論じる．

l ：操作の引数に現れた記号（要素文字列）の数
m ：登録されている記号の数
n ：登録されている名前の数
h ：名前の一覧操作において，対象となるディレクトリが保持している名前の数

これらのパラメータは完全には独立ではない．m 個の記号で表現可能な名前の数は，2^m 個である．逆に，n 個の名前を表現するためには，最低限 $log_2 n$ 個の記号があれば十分である．$m = \log_2 n$ と仮定すると $O(m) = O(\log_2 n)$ となり，$O(m)$ と $O(n)$ では $O(n)$ の方があきらかに大きい．しかし実際には，記号の利用頻度に大きな偏りがあるため，$O(m)$ と $O(n)$ に大きな違いはない．**図 5** は，記号索引に含まれている記号とその名前番号リストの長さの関係を示している．名前番号リストの長さは，その記号を含む名前がいくつあるか，すなわち，記号の出現頻度を表している．この図は，両対数の軸を持つ．このように，記号の出現頻度には大きなばらつきがあることがわかる．それでも n が大きい時，$O(m) < O(n)$ である．よって以下の解析では，簡単のために $O(m)$ の代わりにそれより大きい $O(n)$ を用いる．

SetNS の索引が B+木をつかっているとき，入出力回数のオーダは，**表 4** のようになる．詳しい説明は，付録 A.1 に示す．

このように，キャッシュが無効のとき，名前の検索（解決），登録，および，削除操作において必要な入出力回数は，木構造に基づく名前サービスと比較して B+木の計数 $(\log n)$ 倍の入出力回数が必要である．ただし，名前の数の対数で入出力回数が増加するだけであり，名前の数に関して十分なスケーラビリティを持っているといえる．ディレクトリに対する名前の一覧操作における入出力回数は，木構造と異なり，SetNS の場合そのディレクトリが保持している名前の数 h に大きく依存する．したがって，3.3 節で述べたように，ブラウズするためのコマンドを工夫する必要がある．

表 4 入出力回数のオーダ（B+木により索引付けを行っている場合）
Table 4 The order of the number of I/O when the B+ trees are used for indexing.

Operations	SetNS	Tree-based Name Service
lookup, register and delete	$O(l \log n)$	$O(l)$
list	$O(h \log n)$	$O(l)$

6. おわりに

本論文では，記号の集合に基づく名前サービス SetNS の概念とファイル・システムにおける利用について述べた．SetNS では，名前は，記号（要素文字列）の集合として表される．SetNS の特徴は，1 つの名前を構成する記号を任意の順序で指定可能であること，および，長い名前を省略形で指定可能であることにある．SetNS では，仮想ディレクトリ，コンテキスト，部分名，および完全名といった概念を導入した．SetNS の実現方式の 1 つとして，記号索引と名前番号表と呼ばれる，2 つの索引付けされた二次記憶上の表を用いる方法を示した．

The Open Directory Project が管理している WWW 用のディレクトリ・サービスのデータを SetNS に登録し，必要な二次記憶の量を調べた．その結果，約 31 万個の名前を登録した場合，約 110 M バイトになることが分かった．これは，元の WWW ページを保存するためのデータの約 5%と見積もられる．SetNS の性能は，ディスク入出力回数により決定される．したがって，ディスク・キャッシュが有効なときには性能上の問題はない．ディスク・キャッシュが無効なときの入出力回数は，登録されている名前の数を n，引数で与えられた記号の数を l とすると，検索，登録，削除で，$O(l \log n)$ になる．一覧操作では，その仮想ディレクトリに含まれている名前の数を h とすると，$O(h \log n)$ になる．

今後の課題は，ファイル・システム以外で SetNS に基づく名前サービスを実現することである．今回は，木構造に基づく名前サービスを利用することを想定して作られたデータを元にして，SetNS の名前を合成して実験を行った．今後は，最初から SetNS の機能を

想定して名前を付けた時に記号の分布や名前の長さがどう変化するかを調べてみたい．

参 考 文 献

1) Berkeley DB, Sleepycat Software Inc, New Rider Publishing, http://www.sleepycat. com/ (2001).
2) Gopal, B. and Manber, U.: Integrating Content-based Access Mechanisms with Hierarchical File Systems, *Proc. 3rd Usenix Symposium on Operating Systems Design and Implementation* (*OSDI*), pp.265–278 (1999).
3) Gifford, D.K., Jouvelot, P., Sheldon, M.A. and O'Toole, J.W., Jr.: Semantic File Systems, *Proc. 13th ACM Symposium on Operating System Principles* (*SOSP13*), *Operating System Review*, Vol.25, No.5, pp.16–25 (1991).
4) Neufeld, G.W.: Descriptive Names in X.500, *Proc. ACM Symposium on Communications Architectures & Protocols (ACM SIGCOMM '89)*, pp.64–71 (1989).
5) 大山，千田，戸部，窪田，田中，空：X.500 ディレクトリ入門，東京電機大学出版局 (2001).
6) The Open Directory Project, http://dmoz.org/about.html (2001).
7) Resource Description Framework (RDF) Model and Syntax Specification, *World Wide Web Consortium (W3C)* (1999).
8) Sechrest, S. and McClennen, M.: Blending Hierarchical and Attribute-based File Naming, *12th IEEE International Conference on Distributed Computing Systems*, pp.572–580 (1992).
9) 新城，村松：記号の集合に基づく名前サービスの提案，情報処理学会コンピュータシステム・シンポジウム，Vol.98, pp.9–16 (1998).
10) 新城，西尾，板野：記号の集合に基づく名前サービス SetNS の実現，情報処理学会研究会報告（*SWoPP-2001*）2001-OS-88-8, Vol.2001, No.78, pp.51–58 (2001).
11) Solaris 8 Reference Manual, Sun Microsystems, Inc. (2000).
12) SunOS Network Programming, Sun Microsystems, Inc. (1990).
13) 多田，轟木，樋口：非階層型名前空間を持つ分散ファイルシステム，日本ソフトウェア科学会第 16 回大会論文集，pp.45–48 (1999).
14) Yahoo, http://www.yahoo.com/ (2003).

付 録

A.1 入出力回数

名前の検索（解決）では，記号索引を引数で与えられた記号の回数だけ調べることになる．記号索引は，B+木で構成されている．1 つの記号について 1 回の入出力操作で名前番号のリストを読み込むことが可能であるとすると，入出力回数は，$O(l \log n)$ となる．

名前の登録における入出力回数を考える．名前の登録においては，まず重複した名前を登録しないように，検索が行われる．これは，上で論じたように，$O(l \log n)$ である．次に名前番号表には 1 回の登録が行われるが，これは内部的に B+木が使われているので，$O(\log m)$，すなわち $O(\log n)$ となる．そして，記号索引の更新が行われ，$O(l \log n)$ 回の出力が行われる．$O(\log m)$ と $O(\log n)$ は等しいと仮定したので，よって，全体の入出力回数は $O(l \log n)$ となる．

名前の削除操作における入出力回数のオーダは，名前の登録操作におけるものと同じである．

ディレクトリに対する名前の一覧操作における入出力回数は，そのディレクトリが保持している名前の数 h に大きく依存する．名前の一覧では，まず検索が行われ，ディレクトリ・オブジェクトが作られる．これに要する入出力回数は，$O(l \log n)$ である．一覧操作では，各名前について，名前番号表が引かれる．これにともなう入出力回数は，各名前あたり $O(\log n)$，全体で $O(h \log n)$ となる．それらの名前が持つ記号のすべてについて，OR 演算が行われるが，これはメモリ上で行われるので，入出力は行われない．よって，名前の一覧全体では，入出力回数は，$O((l+h) \log n)$ となる．

h が小さいときには性能的に問題はない．h が大きいとき，l は無視できる．したがって，名前の一覧に要する入出力回数は，$O(h \log n)$ となる．

(平成 15 年 1 月 23 日受付)
(平成 15 年 5 月 20 日採録)

新城　靖（正会員）

1965 年生．1993 年筑波大学博士課程工学研究科電子・情報工学専攻修了．同年琉球大学工学部情報工学科助手．1995 年筑波大学電子・情報工学系講師，2003 年同助教授．分散型オペレーティング・システム，並列処理，情報セキュリティの研究に従事．1995 年情報処理学会山下記念研究賞受賞．博士（工学）．日本ソフトウェア科学会，ACM，IEEE CS 各会員．

西尾　克己　1976 年生．1999 年筑波大学第三学群情報学類卒業．2001 年筑波大学博士課程工学研究科退学．修士（理学）．

板野　肯三（正会員）

1948 年生．1977 年東京大学大学院理学系研究科物理学専門課程単位取得後退学．1993 年より筑波大学電子・情報工学系教授．計算機アーキテクチャ，分散システム，プログラミングシステム等に関する研究に従事．理学博士．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE CS 各会員．