システムソフトウェアと
オペレーティング・システム 88−8
（２００１．７．２７）

記号の集合に基づく名前サービス SetNS の実現

新城　靖[†]　西尾克己[†]　板野肯三[†]

この論文は、新しい名前サービス SetNS (Set Name Service) と、Unix のファイル・システムにおける SetNS の実現について述べている。SetNS では、名前は、記号 (文字列) の集合として指定される。利用者は、任意の順序で記号を与えることができる。さらに、長い名前を省略形で指定することや、共通の記号を持つ名前のグループを一括して扱うことが可能である。Unix の木構造に基づく名前と SetNS の名前を共存させるため、仮想ディレクトリの概念が導入され、新しい名前の構文が定義されている。さらに、SetNS を有効に扱うための新しいシステム・コールとコマンドを示している。

SetNS : A Name Servece based on Set of Symbols and its Implementation

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

This paper describes a new name service called SetNS (Set Name Service) and its implementation in a Unix file system. In SetNS, a name is specified with a set of symbols (character strings). A user can give symbols in arbitrary order. Furthermore, SetNS enables to abbreviate a long name and to simultaneously deal with a group of names that include the same symbols. This paper describes the idea of virtual directories and the syntax of name description for coexistence with the tree-based names of Unix. This paper also shows new system calls and commands for effective use of SetNS.

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、文字列のような人間によって解釈される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・システムにおける名前解決では、利用者から与えられた文字列の名前が、システムの内部で使われる番号や構造体へのポインタへ変換される。

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木構造が使われている。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、もともと木構造である情報を扱う時には特に有効である。要素数が増えた時にも、効率よく名前サービスを実現する手段が知られている。

木構造に基づく名前空間では、名前を付ける時に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、「ネットワークに関係したもの (`net`)」と「コマンド (`net`)」は、独立した概念であるが、木構造で表現する時には、`"net/cmd"`、または、`"cmd/net"`のいずれかの順序を選択しなければならない。この順序が自明ではない時には、ファイルを探すことが困難になる。

多くのファイル・システムでは、階層の順序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・リンクの機能により、DAG 構造となる。上で述べた例では、`"net/cmd"`に対して`"cmd/net"`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。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困難になる。さらに、ファイル名の変更やファイルの削除に対応してシンボリック・リンクを管理することが難しくなる。

このような問題点を解決するために、我々は、木構造とは異なるモデルに基づいた新しい名前サービス SetNS (Set Name Service) の研究を行っている [7]。

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

SetNS 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。

- 名前を構成している記号の順序を入れ替えることができる。木構造では、上下の階層を入れ替えることに相当する。
- 局面に応じた方法で名前を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。
- 共通の記号を持つ名前のグループを一括して扱うことができる。

逆に、SetNS では、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。木構造では、たとえば、`/java/cmd/java` 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が、SetNS ではそのようなことはできない。

文献 7) では、Java により記述された SetNS に基づく名前サーバのプロトタイプについて述べた。そのプロトタイプでは、名前番号表と記号索引という2つの主記憶上の索引付けされたデータ構造を用いて SetNS を実現している。この論文では、SetNS を Unix のファイル・システムで利用可能にする方法について述べる。名前索引と記号番号表は、二次記憶上の索引付けされたデータ構造として実現する。SetNS を Unix のファイル・システムで利用可能にするために、Unix の木構造に基づく名前と SetNS に基づく名前を共存させる方法を提案する。ファイル・システムにおいて SetNS を利用する際の利便性を向上させるために、いくつかのシステム・コールとコマンドを提供する。

2. 関 連 研 究

関連研究については、文献 7) においても論じている。Yahoo[8] に代表されるようなWWWのディレクトリ・サービスでは、HTML のハイパーリンク機能を利用して、部分的に階層の入れ換えを許しているように見せかけている。ハイパーリンクによる疑似的な階層の作成は、完全ではなく、直接 URL を指定して疑似的な階層にアクセスすることはできない。

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換えることを許している[2)5)]。これらのシステムでは、必ず属性と属性値の組を指定する必要がある。本 SetNS では、そのような制約はない。文献 6) 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る。このシステムでは、名前は、左から右へ要素ごとに解釈される。名前の要素が属性に関係している部分では、要素の入れ替えが可能である。SetNS では、名前の要素は大域的であり、全ての名前の要素が入れ替え可能である。

3. 記号の集合に基づく名前サービス SetNS

この章では、名前サービス SetNS の概要、および、さまざまな用語の定義を行なう。

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。**要素文字列**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): 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 番号が、オブジェクト識別子である。名前サービスを提供するプログラムを名前サーバと呼ぶ。ファイルの名前をファイル名と呼ぶ。

SetNS は、名前サービスの一種である。この論文では、SetNS における名前は、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。たとえば、次の文字列は、記号の集合でり、SetNS の名前である。

- `net,cmd,rlogin,`
- `,x11,cmd,xterm,`

SetNS において、名前をグループ化したり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。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。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。たとえば、コンテキストが `",net"`である時、コンテキスト依存の名前 `"cmd,rlogin"`は、コンテキスト独立な名前としては、

",net,cmd,rlogin"を意味する。コンテキスト独立の名前とコンテキスト依存の名前を区別する必要がある場合、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。

名前の操作の引数に現れる名前は、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、名前を構成する全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、高々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。たとえば、次の2の名前が登録されている状況を考える。

- ,net,cmd,rlogin
- ,x11,cmd,xterm

部分名 ",cmd" は、両方の名前にマッチする。一方、部分名 ",xterm" では、後者にのみマッチする。なお同じ ",xterm" という文字列でも、それが完全名の場合には、",x11,cmd,xterm" にはマッチしない。

明示的に完全名と部分名を指定するために、それぞれ:full と :partial という記号を含める。たとえば、",xterm" という名前が完全名であることを示すには、",xterm,:full"、あるいは、",:full,xterm" と記述する。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、部分名が便利である局面が多い。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。

4. Unix のファイル・システムにおける SetNS

3 章で述べた SetNS の考え方は、さまざまな名前サービスで利用可能である。SetNS を Unix のファイル・システムで利用するためには、木構造に基づく名前との共存を計ることや既存のアプリケーションに対する互換性を持たせる必要がある。この章では、Unix のファイル・システムにおいて SetNS を利用するために導入した概念、名前の構文、および、新しく導入したシステム・コールとコマンドについて述べる。

Unix のファイル・システムにおいて用いられている木構造に基づく名前サービスを、この論文では、TreeNS と呼ぶことにする。TreeNS における区切り文字は '/'、TreeNS における名前は、1つ以上の要素文字列を区切り文字'/' で区切ったものである。TreeNS では、要素文字列の順序には意味がある。たとえば、"/cmd/x11" と"/x11/cmd" は、異なる名前である。

4.1 仮想ディレクトリとファイル名の構文

SetNS においては、本来、ディレクトリという考え方は存在しない。木構造に基づく名前サービスとの親和性を高めるために SetNS へディレクトリ (**仮想ディレクトリ (virtual directory)**) の考え方を導入する。SetNS におけるディレクトリは、次の3つの役割を果たす。

(1) コンテキスト（記号の集合）を保持する。
(2) (1) の記号を全て含む名前のリストを保持する。
(3) ディレクトリの内容として、(2) が持つ記号のリストを提供する。ただし、コンテキスト自身に含まれている記号を除く。

たとえば、ディレクトリ ",cmd" は、",cmd,net,rlogin" や",cmd,X11,xterm" のように、名前の一部に"cmd" という記号を含む名前を全て含んでいると考える。ディレクトリ",cmd" の内容は、含んでいる全ての名前を構成している記号を全て集めたものである。たとえば、"net,cmd,rlogin" と "x11,cmd,xterm" という名前があった時には、そのディレクトリの内容としては、次の記号を含む。

net, rlogin, x11, xterm

TreeNS では、プロセスごとにカレント・ワーキング・ディレクトリがあり、相対パス名の解決に用いられる。一方、SetNS では、プロセスごとに**カレント・コンテキスト (ディレクトリ)** を持たせる。そして、TreeNS における絶対パス名には、SetNS のコンテキスト独立な名前、TreeNS における相対パス名には、SetNS のコンテキスト依存な名前を対応させる。カレント・コンテキストは、Unix の応用プログラムからはカレント・ワーキング・ディレクトリとしてアクセスすることを可能にする。

仮想ディレクトリ、および、コンテキスト・ディレクトリの概念の導入により、次のようなコマンドやシステムコールが SetNS に基づく名前についても利用可能になる。

- ls コマンド、getdents(get directory entries) システムコール
- chdir コマンド、chdir システムコール

仮想ディレクトリは、名前を登録することで暗黙的に作られる。よって、mkdir や rmdir のようなコマンドやシステムコールで明示的にディレクトリを操作することは許されない。

TreeNS と SetNS を共存させるとは、Unix のシス

```
<name> ::= <tree_name> <set_name>
<set_name> ::= <cindep_name> | <cdep_name>
<cindep_name> ::= "," <cdep_name>
<cdep_name> ::= <string> {","<string>|<special>}
<tree_name> ::= <abs_name> | <rel_name>
<abs_name> ::= "/" <rel_name>
<rel_name> ::= <string> {"/" <string>}
<special> ::= ":full" | ":partial"
<string> ::= "/"と","を含まない文字列
```

図 1 SetNS に基づく名前と木構造に基づく名前の共存させるための新しいファイル名の構文。
Fig. 1 The new syntax of file names for coexistence of SetNS-based names and tree-based names.

テム・コールのレベルでは、引数として名前を取るものについて、従来の TreeNS の名前と新しい SetNS の名前の両方を利用可能にすることである。そのようなシステム・コールのうち最も重要なものは `open()` である。

TreeNS と SetNS を共存させるために、Unix のパス名の解釈を変更した。新しく定義した名前の構文の概略を、図 1 に示す。SetNS の名前は、区切り文字として','を含む。TreeNS の名前は、区切り文字として'/'を含む。','と'/'の両方が含まれている時には、先に現れたものを採用する。区切り文字を含まないものは、TreeNS の名前としてあつかう。SetNS の名前として１つしか記号を含まないものを使う時には、たとえば`"sym1,"`のように、末尾に','を付ける。

3 章で述べたように、SetNS では、','で始まる名前は、コンテキスト独立な名前として扱い、それ以外は、コンテキスト依存の名前として扱う。SetNS の名前の中で、`:full` という記号が現れた時には、その名前が完全名であることを意味する。`:partial` は、その名前が部分名であることを意味する。

4.2 SetNS のために導入・変更したシステム・コール

4.1 節で述べたように、各プロセスは、SetNS のコンテキスト依存の名前の解釈のためにカレント・コンテキストを保持する。カレント・コンテキストを変更するためには、既存のシステムコール `chdir()` を用いることにした。これにより、既存の Unix のシェルを一切変更することなく、`chdir` コマンドを使って対話的に SetNS の名前空間をブラウズすることが可能になる。

SetNS のために、カレント・コンテキストを提供するシステム・コール `getcontext()` を新たに導入した。そのインタフェースを以下に示す。

```
int getcontext( char *buf, size_t size )
```

`buf` には、コンテキストが返される。`size` は、`buf` のバイト単位の大きさである。

既存のシステムコール `getdents()` を、SetNS の仮想ディレクトリに対しても働くようにした。`getdents()` は、ディレクトリに含まれている内容を返すものであり、`ls` コマンドから使われている。4.1 節で述べたように、仮想ディレクトリは、コンテキストにマッチした名前のリストを、**名前番号**と呼ばれる、登録されている名前と１対１に対応する番号を使って保持している。`getdents()` では、まずその名前番号のリストから名前の記号の表現を求める。そしてその記号の和集合を計算し、ユーザ・プロセスに返す。

3 章で述べたように、SetNS では、部分名と完全名がある。指定された部分名から完全名を得るためのシステム・コール `getfullname()` を新たに導入した。そのインタフェースを以下に示す。

```
int getfullname( char *name,
      namelist_t *buf, int bytes )
```

`name` は、完全名を調べたい部分名である。結果として、複数の完全名が返されることがある。`namelist_t` は、`getdents()` の結果を返すために使われる `dirent` 構造体と類似の構造体である。

4.3 コマンドレベルでの SetNS の利用

従来の Unix のコマンドの多くは、そのままで SetNS の名前で指定されたファイルを読み書きすることができる。たとえば、次のような Unix の基本的なコマンドは、ディレクトリ木を扱う場合（再帰的な処理を伴う場合）をのぞいて、利用可能である。

`ls, cp, mv, rm`

SetNS 独自の機能を活用するために、以下のようなコマンドを追加した。

(1) `getcc` (get current context)

プロセスごとのカレント・コンテキストを表示するために `getcc` コマンドを提供する。これは、従来の `pwd` (print working directory) コマンドに相当する。このコマンドは、内部的には `getcontext()` システム・コールを実行している。

(2) `snren` (SetNS Rename)

TreeNS では、ファイルの名前を変更する場合、`mv` (move) コマンドが使われる。これは、コマンド・ラインの引数で与えられた名前を、`rename()` システム・コールに渡している。SetNS の名前を `mv` コマンドで変更する時、その名前が完全名だった時には何の問題も生じない。しかし、その名前が部分名だった場合に、引数に現れていない記号が失われるという問題が

ある。

この問題を解決するために、新たなコマンド snren を導入した。このコマンドは、まず、与えられた名前にマッチする全ての名前の完全名を、4.2 節で述べたシステム・コール getfullname() で求める。次にこの完全名に対して、与えられた引数から改名後の完全名を求める。最後に、完全名を指定して、rename() システム・コールを実行する。

(3) addsym と delsym (add/delete symbol)

SetNS では、名前を整理する場合、特定の記号を追加すること、および、削除することが頻繁に行われる。その作業を支援するために、記号を追加するコマンド addsym と削除するコマンド delsym を提供する。これらコマンドは、snren コマンドと同様に、それぞれの名前について部分名から完全名を求め、引数で指定された記号を付加、あるいは削除する。

(4) copy-all と remove-all

木構造に基づく名前では、ある部分木全体をコピーする操作や削除する操作がある。Unix では、それぞれ cp コマンドや rm コマンドに -r (recursive) オプションを与えることで、そのような操作が可能になっている。一方、SetNS では、ある共通の記号を持つ名前のファイルを一括して扱う場合、再帰的なアルゴリズムを使うことができない。それは、仮想ディレクトリを再帰的に手繰った場合、同じ名前の一部分が繰り返し現れるからである。

頻繁に利用されるコマンド cp と rm については、特別のコマンド copy-all, remove-all を提供する。これらは、指定された部分名にマッチした名前を持つファイルを全てコピーする、あるいは、全て削除するコマンドである。内部的には、copy-all コマンドは、snren コマンドと同様に完全名を得て、rename() システム・コールを発行する代りにコピーを行う。remove-all コマンドは、完全名を引数として unlink() システム・コールを発行する。

5. システムコール・トレースによる SetNS の実現

4 章で述べた Unix のファイル・システムにおける SetNS を、世界 OS5[3] と同様に、System V 系の Unix (Solaris) が持つシステム・コールのトレース機能を用いて実現した。システム・コールのトレース機能では、対象となるプロセスを指定されたシステム・コールの実行直前、あるいは、直後で停止させることができる。そして、システム・コールの引数や結果を書き換えることができる。

現在、SetNS の名前を持つファイルを、特定のディレクトリの下にシリアル番号を付けて保持している。ここでは、そのディレクトリを"/setns/"とする。システム・コールの引数に、4.1 節で示したような SetNS の名前が現れると、そのディレクトリの下のファイル名に書き換える。たとえば、",a,b,c"という SetNS の名前を、"/setns/N"書き換える。N は、唯一になるように、シリアル番号から作成した文字列である。

4.2 節では、新たにいくつかのシステム・コールを導入した。このようなシステム・コールは、実行環境である Solaris[4] には存在しない。システム・コールのトレース機能は、既存のシステム・コールに対してのみ働き、存在しないシステム・コールについてはエラーになってしまう。そこで、新たなシステムコールを実現するために、readlink() システム・コールを利用した。readlink() は、シンボリック・リンクの内容を読み込むためシステム・コールであり、次のようなインタフェースを持つ。

```
int readlink(const char *path, char *buf,
        size_t bufsiz)
```

このシステム・コールにおいて、パス名として特別な文字列が指定された時には、特殊な処理を行なうことで新たなシステム・コールを実現する。たとえば、path が ":getcontext" ならば、getcontext() システム・コールであると解釈し、その結果を buf に保存する。

SetNS サーバの内部で使われているアルゴリズムは、文献 [7] で示したプロトタイプとほとんど同じである。そのアルゴリズムは、**名前番号表**と**記号索引**という 2 つの索引付きのデータ構造を用いる。プロトタイプでは、Java 言語で標準的に利用可能なメモリ中のハッシュ表を用いた。Solaris での実現では、これらの表を Berkeley DB[1] により索引付けを行ない、二次記憶上に保存した。

6. SetNS の利用例:マニュアルの管理

ファイル・システムにおける SetNS の有効性を確認するために SetNS の機能を活用したマニュアル表示プログラムを作成した。その結果として、既存のプログラムである man コマンドにはない機能を簡単に提供することができること、対話的な利用では十分な性能を持つこと、および、必要な二次記憶容量が実用的な範囲に収まっていることを示す。

6.1 SetNS におけるマニュアルの名前付け

Solaris における約 7700 個のマニュアル・ファイルを、SetNS の機能を利用して以下のように名前を付

けた。

SetNS における名前付けでは、まず、共通に利用する記号の集合を決定する。個々のファイルの名前を決定する時、各記号について意味を考慮して、そのファイルがその記号の意味と適合すれば、その記号をそのファイルの名前に含める。たとえば、cmd という記号は、コマンドのマニュアルを保存するファイルに付ける。これは、Solaris では、1 章と 8 章の項目に付ける。また、network という記号は、コマンド、システム・コール、ライブラリのいずれの場合でも、ネットワーク関連であれば付ける。API という記号は、システム・コール、ライブラリなどプログラミングに必要な項目につける。

たとえば、ライブラリ関数 socket() のマニュアルは、Solaris では次のファイルに保存されている。

/usr/share/man/sman3socket/socket.3socket

これを、SetNS では、次のような名前のファイルに保存する。

,API,lib,man,network,sgml_format,socket

6.2 sman コマンド

6.1 節で述べた方法に従い名前を付けたファイルを表示するプログラム sman を作成した。その典型的な利用例を以下に示す。

```
% sman write
API cmd dev sgml_format sys_call usr
% sman write,sys_call
(write システム・コールのマニュアルを表示)
% _
```

この例では、write システム・コールのマニュアルを表示しようとしている。しかし、write という名前でヒットするファイルは、複数存在し、1つに特定することができない。よって sman コマンドは、ファイルの内容を表示する代りに、次にしぼり込むために使えるキーワードを表示している。利用者は、次に write,sys_call という名前を与えている。これでファイルが特定されたので、sman コマンドは、その内容を表示する。

図 2 に、sman コマンドのプログラムの概略を示す。このように、SetNS の機能を利用することより、非常に単純なプログラムで sman コマンドの機能が実現されていることがわかる。このプログラムは、まず、与えられた引数に "man," という記号を付け、stat() システム・コールを実行し、ファイルの型を調べている。複数の名前が見付かった時には、仮想ディレクトリが作成され、ディレクトリ型が返される。その場合は、ディレクトリの内容を表示する。名前が1つだけ見付かった時には、ファイル型が返される。その場合は、そのファイルの内容を表示している。

```
main( int argc, char *argv[] ){
  char name[MAXMANFLEN];
  struct stat buf;
  strcpy( name, "man," );
  strcat( name, argv[1] );
  if( stat( name, &buf ) == 0 ) {
    if( S_ISDIR(buf.st_mode) )
      list_directory( argv[1] );
    else
      show_file( argv[1] );
  }
  else{
    perror( argv[1] );
    exit( 1 );
  }
}
```

図 2 SetNS の機能を利用する sman コマンドの概略
Fig. 2 The overview of the sman command that uses the SetNS functionality.

6.3 実　　験

Solaris 8 の /usr/share/man 以下に保存されているマニュアルについて、SetNS の機能を利用して名前を付けた。そして、二次記憶上のデータの大きさと処理時間を調べた。利用した計算機は、Sun Enterprise 450 である。CPU 数は、4、各 CPU は、Ultra SPARC II 480 MHz、キャッシュの容量は 8M bytes、共有メモリは 2G bytes である。

まず、名前を保存するために必要な二次記憶上のデータの大きさを調べた。この実験では、Sloaris 8 のマニュアル約 7700 個のファイルを、6.1 節で述べた方法で名前を付けて登録した。ファイルの大きさは、全体で約 50M バイトであった。SetNS では、名前番号表と記号索引という、B＋木で索引付けされた二次記憶上のデータ構造を用いる。ファイルの大きさは、合計で約 2M バイトであった。これは、ファイルの大きさの合計約 50M バイトの約 4% に相当する。これは、ディスクの大容量化と低価格化が進んでいる現在、実用上まったく問題がないと言える。

次にシステム・コールの実行時間を測定した。まず、名前の検索のみを行なう stat() システム・コールについて実行時間を調べた。1個から3個の記号を含む名前の場合、キャッシュがヒットする状態で1回あたり 0.6m 秒から 2m 秒であった。この時間は、現在広く使われているディスクのシーク時間 (5m 秒程度) や

回転待ち時間 (5m 秒程度) と比較しても小さい。よって実行時間はディスク入出力回数により決定される。

名前の検索におけるディスク入出力回数を評価する。登録されている名前の数を n、検索操作の引数で指定された名前が含んでいる記号の長さ l とする。名前の検索では、記号索引を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、ディスク入出力回数は、$O(l \log n)$ となる。一方、木構造に基づく名前サービスにおける入出力回数は、要素文字列の数を l とすると、$O(l)$ となる。従って、木構造に基づく名前サービスと比較して、SetNS の検索の性能は、対数関数の係数だけ重たい。ただし、対数関数であるので、スケーラビリティに関しては実用上問題がないと思われる。

次に名前の登録を含むシステム・コール `open()` の実行時間を測定した。これには、上でのべた Solaris 8 のマニュアルを用い、何も登録されていない状態から 7700 個の登録が完了するまでの全体の時間を測定した。その結果、システム・コール1回あたりの実行時間は、平均で 23m 秒であった。この絶対時間については、対話的な利用においては実用上十分な性能が得られたと言える。なお、Unix において1つのディレクトリで同数のファイルを作成した場合、平均 13m 秒であった。これからシステム・コールのトレースを含めた SetNS のオーバヘッドは約 10m 秒であることがわかる。

名前の登録におけるディスク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、検索が行なわれる。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回の登録が行なわれるが、これは内部的にB＋木が使われており、$O(\log n)$ となる。よって、最終的には、検索と同じ $O(l \log n)$ となる。

仮想ディレクトリに対する一覧操作は、木構造に基づく名前サービスよりも重たい。一覧操作は、`ls` コマンドがシステム・コール `getdents()` を通じて利用されている。`ls` の実行時間は、表示される記号の数が 700 個のもの (`cmd,usr`) で 0.5 秒、1300 個のもの (`cmd,`) で 1.3 秒、3300 個のもの (`lib,`) で 7 秒であった。

このように、実行時間は、その仮想ディレクトリが含んでいる名前の数（この場合、記号の数とほぼ等しい）に大きく依存する。この仮想ディレクトリが含んでいる名前の数を m とする。マニュアルの場合、`cmd` や `lib` のように、多くの名前で共通に使われる記号の場合、たくさんの名前がヒットし、m が大きくなる。一方、`fork` のように、特定のシステム・コールの名前では、m は、小さくなる。`getdents()` では、仮想ディレクトリが含んでいる記号の一覧を作成する時に、記号の集合の和集合を計算している。この和集合の計算には、現在単純なアルゴリズムが使われており、その計算量は、おおよそ $O(m^2)$ になっている。

一覧操作の入出力回数を考える。まず、検索と同様に与えられた名前から、それらの記号を名前の要素としてもつ名前が全て求められる。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。これに伴う入出力回数は、各名前あたり $O(\log n)$、全体で $O(m \log n)$ となる。それらの名前が持つ記号の全てについて、和集合が求められるが、これはメモリ上で全ての処理は行なわれるため入出力は行なわれない。よって、名前の一覧全体では、入出力回数は、$O((l+m) \log n)$ となる。m が小さい時には、性能的に問題はない。m が大きい時、l は、無視できるので、結局、名前の一覧に要する入出力回数は、$O(m \log n)$ となる。すなわち、名前の数と全ての名前の数の対数に比例する。

名前の数 m が大きい場合、入出力回数のオーダーは、木構造に基づく名前サービスものと比較して大きい。Unix では、1つのディレクトリの内容を読み込むために必要な入出力回数は、$O(m)$ であり、全体の名前の数 n から独立している。

7. 制　　限

今回は、Unix のファイル・システムにおいて SetNS を実現した。現在の実現では、いくつかの制限がある。この章では、そのような制限を、由来ごとにまとめる。

まず、SetNS そのもの制限がある。木構造に基づく名前サービスと比較して、1章と2章で述べたように、SetNS では、記号の重複が許されていない。従って、木構造に基づく名前サービスでは表現可能な名前が、SetNS では表現できないことがある。

次に、SetNS と木構造に基づく名前サービス TreeNS と混在させたことによる制限がある。まず、今回の方法では、1つの名前の中で SetNS と TreeNS の名前を混在させることはできない。また、区切り文字として','を使用したことにより、それを含むファイル名が TreeNS では、利用できなくなった。TreeNS では、あるディレクトリを根とする部分木の操作が可能である。この時、再帰的なアルゴリズムが利用される。一方、SetNS では、共通の記号を持つ名前のグループ

をまとめて扱う時には、再帰的なアルゴリズムを使うことができない。この問題点を補うために、SetNS では、サービスのレベルで部分名から完全名への変換、システム・コールのレベルでは、`getfullname()` を提供する。

既存の Unix の応用プログラムを利用する上で、独自にファイル名を生成するプログラムの利用に制限がある。たとえば、emacs は、木構造を仮定して名前を生成する機能がある。そのようなプログラムは、SetNS の名前をうまく扱えない。

最後に、システム・コールのトレースを用いて実現したことによる制限がある [3)]。それは、実行速度が低下することと、および、デバッガなどの他にトレースを利用しているプログラムと競合してしまうことである。

その他に、残された課題としては、次のようなものがある。

(1) 複数の空間。Unix では、複数の木構造を実現するために、`chroot()` システム・コールが用意されている。また、あるディレクトリ以下の部分木を一つの空間と考えることも可能である。SetNS では、現在の所、１つのプロセスで１つの空間しか扱えない。

(2) アクセス制御。木構造では、部分木単位でアクセス制御が可能である。現在の SetNS の実現では、空間全体が１つのアクセス・モードで管理されている。記号を用いたアクセス制御の仕組みが必要である。

(3) シンボリック・リンクの扱い。現在は、SetNS では、シンボリック・リンクの意味に不明な部分があるので利用していない。たとえば、`lib -> library` のようなリンクは、シソーラスと考えれば、問題はない。しかし、`x -> a,b` のようなリンクを許せば、`x,a` のようなリンクの意味が不明である。

8. おわりに

本論文では、記号の集合に基づいた名前サービス SetNS について述べた。その特徴は、複数の記号の順序を入れ替えることができること、長い名前を省略形で指定可能であること、および、共通の記号を持つ名前のグループを一括して扱えることにある。

本論文では、SetNS を、Unix のファイル・システムにおける名前サービスとして利用する方法について述べた。そのために、仮想ディレクトリとカレント・コンテキストという概念を新たに導入した。さらに、木構造に基づく名前と SetNS の名前を共存させるための名前の構文を示した。SetNS の機能を有効に利用するためにいくつかのシステム・コールとコマンドを導入した。これらの機能を、Solaris においてシステム・コールのトレースを用いて実現した。二次記憶上のデータ構造は、Berkeley DB を利用してＢ＋木により管理される。

実現したシステムを利用して、Solaris 8 に含まれているマニュアルを表示するプログラムを作成した。そして、しぼり込み検索などの機能が簡単に実現できることを示した。実現したシステムの性能は、名前の操作のうち、検索、登録、削除は入出力回数で決定されることを示した。それらの名前の操作に必要な入出力回数は、登録されている名前の数の対数にも影響される。名前の一覧操作におけるメモリ中で行われる和集合の計算には、改善の余地がある。

今後の課題は、名前の一覧操作を高速化することである。また、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したい。さらに、SetNS に基づくファイル名を簡単に扱えるためのファイル選択ダイアログなどを開発したい。

参 考 文 献

1) “Berkeley DB Version 3”, `http://www.sleepycat.com/` (2001).
2) D.K.Gifford, P.Jouvelot, M.A.Sheldon, and J.W.O’Toole,Jr. : “Semantic File Systems”, Proc. 13th ACM Symbpsium on Operating System Principles (SOSP13), Operating System Review, Vol.25, No.5, pp.16-25 (1991).
3) 石井, 新城, 西尾, 孫, 板野: ”並列世界モデルに基づくＯＳのシステムコールトレースによる実現”, 情報処理学会研究会報告 99-OS-81-15, Vol.99, No.81, pp.83-88 (1999 年 5 月).
4) “Solaris 8 Reference Manual”, Sun Microsystems, Inc. (2000).
5) G. W. Neufeld: “Descriptive Names in X.500”, In Symposium Proceedings on Communications Architectures & Protocols (ACM SIGCOMM ’89), pp.64-71 (1989).
6) S. Sechrest and M. McClennen: “Bendig Hierarchical and Attribute-based File Naming”, IEEE 12th International Conference on Distributed Computing Systems, pp.572-580 (1992).
7) 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 年 11 月).
8) Yahoo, http://www.yahoo.com/ (2001).