システムソフトウェアと
オペレーティング・システム 81－15
（１９９９．５．６）

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

石井 孝衛, 新城 靖, 西尾 克己, 孫 軍, 板野 肯三

並列世界モデルとは、世界とよばれるプロセスの実行環境の動的な生成、削除、融合を許すモデルである。このモデルの応用には、ソフトウェアの安全なインストールや高度なアクセス制御がある。この論文は、Unix System V が提供しているシステムコールのトレース機能を用いて並列世界モデルを実現する方法について述べている。この方法では、カーネル外のサーバが、応用プロセスのシステムコールをトレースし、子世界の中で行われた名前の操作の表を維持し、必要に応じてシステムコールの引数に含まれているファイル名を書き換える。

The Implementation of an Operating System based on the Parallel World Model by Tracing System Calls

KOUEI ISHII, YASUSHI SHINJO, KATSUMI NISHIO, JUN SUN, and KOZO ITANO

The parallel world model is a model that allows creating, deleting and merging multiple execution environments of processes called "worlds". The application of this model includes safe installation of software and advanced access control. This paper presents the implementation of an operating system based on this model by using the trace facility of system calls in Unix System V. In this implementation, the server outside the kernel traces the system calls issued by the application processes, maintains tables of name manipulation in child worlds, and rewrites the filenames in the arguments of these system calls.

1. はじめに

仮想計算機システムでは、１つの物理的な計算機上で複数のハードウェア・レベルの実行環境が提供される。マイクロカーネルに基づくシステムで複数のオペレーティング・システムを同時に実行させた場合、それらのシステムにより、全体で複数の実行環境が提供される。いずれのシステムでも、実行環境間では、ファイルやプロセスなどの共有はあまり行なわれない。また、環境間の演算も定義されていない。

我々は、**並列世界モデル (the parallel world model)** に基づくオペレーティング・システムを開発している。[5]。このシステムは、**世界 (world)** と呼ばれるアプリケーション実行環境を複数提供する。世界は、ファイルやプロセスを入れる箱である。世界には、生成・削除・融合という操作が定義されており、世界の内容は、継承や融合により共有される。

世界は、隔離の単位であると同時に、継承の単位でもある。利用者は、ファイルの更新を隔離したい時に、既存の世界を親世界として新しく子世界を**生成**する。この時、子世界の内容は空であるが、継承により親世界の内容を参照することができる。子世界で更新されたファイルは、子世界のみで有効であり、親世界のファイルは変更されない。子世界の内容を親世界において有効にしたい時には、子世界を親世界に**融合**する。この場合、子世界で更新されたファイルは、親世界の対応するファイルを上書きする。子世界の内容を破棄したい時には、子世界を**削除**する。この場合、その世界の内容であるファイルやプロセスも削除される。

並列世界モデルの応用には、次のようなものがある。

(1) ソフトウェアの安全なインストール：子世界でインストールを行ない、動作確認の後に親世界に融合する。大きな問題があった時には子世界を削除する。

(2) セキュリティと情報生存能力の向上：攻撃によりファイルが破壊された時には、その世界を削除することで、迅速に回復させる。

(3) 高度なアクセス制御：特定の世界において、ファイルのモードだけでなくファイル名のパタンや、通信先、通信内容を利用してアクセス制御を行なう。

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

並列世界モデルを実現するためには、次のような機能を実現する必要がある。

- 同名のファイルでも、必要に応じて世界ごとに異なる実体にマップする(2次元の名前空間)。
- ある世界に属するファイルやプロセスを全て削除したり別の世界に移したり一覧を返したりする。

これらの機能を実現する方法として、我々は、これまでに名前解決モジュールを置き換える方法、および、名前解決の後にマッピングをファイルの実体レベルで行う方法を提案してきた[4)5)]。この論文では、Unix System V が提供しているシステムコールのトレース機能を用いる方法を提案する。システムコールのトレース機能とは、システムコールの処理の直前、あるいは、直後にプロセスを停止させたり、システムコールの引数、結果、および、対象プロセスのメモリの内容を読み書きすることを可能にするものである。トレースを用いる方法の利点は、カーネルに一切手を加えずにカーネル外のサーバ・プロセスにより並列世界モデルを実現することができることである。

この論文で述べる実現方法の特徴は、システムコールの引数としてファイルの名前を取るものに着目し、その引数を書き換えることで2次元の名前空間を実現している点にある。あるプロセスがファイルを開くシステムコールを発行すると、それをトレースしているカーネル外の**世界OSサーバ**と呼ばれるプロセスにより捕捉される。そして必要に応じてシステムコールの引数に含まれているファイル名が書き換えられる。この方法の利点は、ファイルを開く時にオーバーヘッドがかかるが、読み書きについては一切のオーバーヘッドがかからないという点にある。

この論文では、従来の3つの世界の操作(生成、削除、融合)に加えて、新たに、「内容を得る」という操作を提案する。これは、対話的に世界を利用する時に有用となる。たとえば、応用(1)では、世界を融合する前にどのようなファイルが更新されるかをこの操作により事前に知ることができる。応用(2)では、侵入者により破壊されたファイルや残されたプロセスを調べることができる。

2. 関連研究

本論文で述べる実現方法は、Apertos システム[6)]で用いられている**リフレクション**(あるいは、メタオブジェクト・プロトコル)を用いる方法や、Fluke システム[3)]で用いられている **nester** を用いる方法と類似している。これらのシステムと比較して、本実現方法の特徴は、世界の4つの操作を提供していること、ファイル名に着目していること、Unix という広く使われているシステムを対象としていることにある。

文献2)では、楽観的処理を支援するために、V System に encapsulation と呼ばれる仕組みを導入する方法が提案されている。encapsulation は、世界と同様の生成・削除の機能があり、融合に相当する必須化という機能がある。encapsulation と比較して世界の特徴は、部分的な融合や、直列化可能性を要求しない応用にも対応することができる点にある。またこの論文で述べる実現方式の特徴は、すべてカーネル外のプロセスで実現できることにある。

3. システムコールのトレースによる並列世界モデルの実現方法の概要

1章で述べたように、並列世界モデルを実現するためには、2次元の名前空間と、世界の4つの操作を実現することが必要である。この章では、これらの機能を実現する方式として、プロセスが実行するシステムコールをトレースし、その引数に現れるファイル名を書き換える方法の概要を述べる。

3.1 システムコールのトレース

本論文では、Unix System V が提供している**システムコールのトレース機能**を利用して並列世界モデルを実現する方法について述べる。この方法の利点は、カーネルに一切手を加えずにカーネル外で動作するプロセスにより並列世界モデルを実現することができることである。今回の実現では、次のような機能を利用した。

- 指定したシステムコールについて、カーネル内での実行の直前、あるいは、直後にトレース対象のプロセスを停止させる。
- システムコールの引数や結果を読み書きする。
- 対象プロセスのメモリの内容を読み書きする。
- システムコールの実行を中止させる。

本実現方法では、特定のシステムコールの引数に含まれるファイル名を書き換える必要がある。そのようなシステムコールについて実行前に止め、書き換える部分の内容を保存し、書換えを行ない、システムコールを実行させる。そして実行後に、保存していた内容を戻す。

この方法の限界として、システムコールのトレース機能を利用する他のアプリケーションを利用できないという点がある。例えば、デバッガやシステムコールを表示するコマンド(truss)を利用することができない。

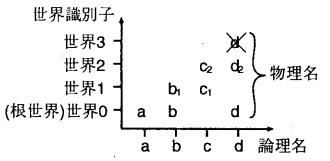

図1　論理名と物理名による2次元名前空間

3.2　ファイルシステムにおける2次元の名前空間

2次元の名前空間とは、ファイルの名前と世界識別子を与えると、ファイルの実体を返すものである。ファイルの実体とは、例えばUnixのカーネル中ではinode (または vnode) である。Unixの namei() と呼ばれる関数は、ファイル名だけから inode を返す。文献4) において、我々は、既に namei() を置き換える形でファイル名とプロセスが属している世界の識別子から inode を返す方法を提案している。しかしながら、カーネルの外で並列世界モデルを実現しようとした場合、カーネル中のオブジェクトである inode を用いることはできない。この問題を解決するために、次の2種類のファイル名を用いる。

(1)　**論理名**：利用者プロセスがシステムコールの引数で指定するファイル名。

(2)　**物理名**：ファイルの実体と1対1に対応したファイル名。

この2つのファイル名を使って2次元の名前空間を実現した様子を図1に示す。この図で、世界0は、**根世界 (the root world)** であり、論理名は、そのまま物理名にマップされる。それ以外の世界では、必要に応じて別の物理名にマップされる。この図では、世界1が根世界の子世界であったとする。

まず、あるプロセスが世界1で論理名"b"のファイルを開いた場合を考える。

```
open("b",O_RDONLY);
```

世界1には、論理名"b"に対応するファイルの実体が存在し、その物理名は"b1"である。よって、その実体をアクセスさせるようにするためには、システムコールの引数を、次のように書き換えればよい。

```
open("b1",O_RDONLY);
```

次に世界1において、論理名"a"のファイルを読み込み専用で開いた場合を考える。

```
open("a",O_RDONLY);
```

世界1には、論理名"a"に対応するファイルの実体が存在しない。この場合、祖先の世界が検索される。そして根世界である世界0に実体が存在することがわかる。今の場合、読み込み専用で開くので、ファイルは書き換えられることはない。よってそのまま根世界にあるファイルの実体をアクセスさせればよい。そのためには、システムコールの引数を書き換えずにそのまま実行させる。

最後に、世界1において、論理名"a"のファイルを書き込み専用で開いた場合を考える。

```
open("a",O_WRONLY);
```

世界1には、論理名"a"に対応するファイルの実体が存在しない。この場合、祖先の世界が検索される。そして根世界である世界0に実体が存在することがわかる。今の場合、書き込み専用で開くので、根世界のファイルの実体が書き換えられないようにコピーを作成する。このコピー先のファイルの実体の名前を"a1"とする。このファイルの実体をアクセスさせるようにするためには、システムコールの引数を、次のように書き換えればよい。

```
open("a1",O_WRONLY);
```

2次元の名前空間には、名前を登録したという情報に加えて、名前を削除したという情報も維持する必要がある。たとえば、図1において、世界3では、論理名"d"が削除されている。この世界が親世界へ融合された時には、その名前は親世界で削除される。

なお、根世界以外のファイルやディレクトリは、全て各パーティションごとに設置するディレクトリの下に作成する。この下にくる名前は、シリアル番号などを用いてユニークになるようにする。

3.3　ファイルシステムにおける世界の操作

ファイルシステムにおける世界の削除と融合とは、その世界に属しているファイルを全て削除する、あるいは、別の世界に移動することに相当する。このためには、まずある世界からその内部に含まれているファイルを得ることができるようにする。カーネルの外でこれを実現するためには、3.2で述べたようにファイル名の操作で実現することになる。

たとえば、図1で、世界1を削除することを考える。このためには、次のようにこの世界に属するファイルの物理名を全て削除すればよい。

```
unlink("b1");
unlink("c1");
```

図1で、世界1を根世界に融合するためには、次のように、根世界のファイルの論理名を削除し、世界1に属するファイルの物理名を論理名に変える。

```
unlink("b");
rename("b1","b");
unlink("c");
rename("c1","c");
```

ファイルシステムにおける世界の内容を得る操作を

実現するためには、その世界に属している全てのファイルの名前のリストを生成する必要がある。たとえば、図1で、世界1については、"b"と"c" というように、世界1に含まれているファイルの実体の物理名ではなく、論理名を返す必要がある。これを実現するには、世界の削除と同様に、ある世界について全てのファイルの実体を探し、その論理名を求める。

3.4 プロセスの扱い

プロセスの場合、ファイルのようにcopy-on-writeを行わないので、異なる世界の間でのプロセス間通信(シグナルを含む)だけが問題となる。これまでは、融合されるまで遅延させることにしていた[5]。たとえば、パイプに対する読み書きが行われた場合、同一世界に属するものについては、そのまま実行させ、異なる世界に属するものについては、融合されるまで遅延させる。今回の実現では、対話的な応用を主に考えているので、疑似端末入出力やTCP/IPによる通信を遅延させずに実行させることにする。

4. 世界OSサーバの実現

3章では、システムコールのトレースによる並列世界モデルの実現の概要について述べた。この章では、**世界OSサーバ**と呼ぶ、カーネル外のプロセスについて述べる。世界OSサーバは、世界間の関係を維持するモジュールを含む。このモジュールは、たとえば、世界の親子関係を調べ、祖先の世界のリストを返す手続きを提供する。この実現方式については、既に提案している方法を用いる[5][4]。この章では、システムコールを捕捉する方法、ファイルシステムについての2次元の名前空間を実現する方法について述べる。

4.1 システムの構成

本システムの構成を、図2に示す。本並列世界モデルに基づくオペレーティング・システム(世界OS)は、世界OSサーバとUnixカーネルから構成される。利用者は、**世界シェル**を通じて世界を操作し、世界を指定してプロセスを作成する。利用者は、子世界に作成したプロセスと、疑似端末やXウインドウの機能を通じて対話を行なう。世界シェルは、世界の操作や世界を指定したプロセスの実行などの要求を、世界OSサーバに送る。一般のプロセスは、普通のUnixのコマンドから作られるものであり、Unixのカーネルに対してシステムコールを発行する。世界OSサーバは、このシステムコールを選択的に捕捉し、引数を書き換えたり結果を書き換えたりする。

4.2 システムコールの捕捉

世界OSサーバは、主に次の種類のシステムコールを捕捉する。

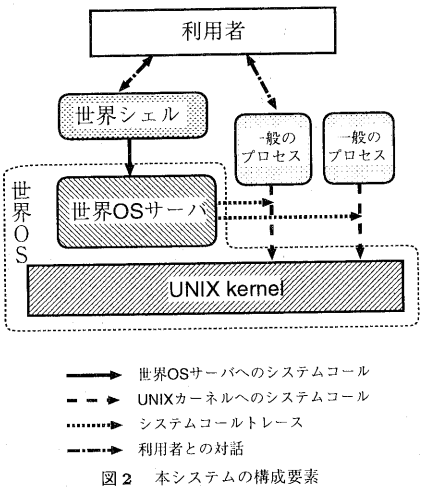

図2 本システムの構成要素

- ファイル名を引数にとるシステムコール。詳しくは、4.4節で述べる。
- **fork()** システムコールとプロセスの終了☆。それぞれ、プロセスを管理する表のエントリの登録と削除を行なう。
- ディレクトリの内容を返すシステムコール **getdents()**☆☆。詳しくは、4.4節で述べる。

このように、全てのシステムコールではなく、特定のものだけを選択的に捕捉している。これにより、**read()** や **write()** など捕捉していないものについては、オーバヘッドがなく実行することができる。

4.3 世界名前操作表とオブジェクト表による2次元の名前空間の実現

3.2節で述べた2次元の名前空間は、**世界名前操作表**と**オブジェクト表**により実現される。図3に、それらの表の例を示す。この表には、次のような情報が保存されている。

- 識別子**wid1**の世界で論理名**"/home/user1/file1"**が登録された。物理名は、**"/home/.world/2"**。
- 識別子**wid1**の世界で論理名**"/home/user1/dir1"**が登録された。物理名は、**"/home/.world/3"**。
- 識別子**wid1**の世界で論理名**"/home/user1/file3"**が削除された。

世界名前操作表は、子世界で行われた「名前の操作」を保持するものである。名前の操作(op)には、登録

☆ プロセスの終了は、exit()システムコールを捕捉するのではなく、独自の機能で捕捉している。

☆☆ ls コマンドが使用している。システムによっては、getdirentries() というシステムコールがある。

世界名前操作表

world	dir_oid	basename	op	oid	op_time
wid1	oid1	"file1"	ADD	oid2	1:00
wid1	oid1	"dir1"	ADD	oid3	1:30
wid1	oid1	"file3"	DEL		2:00

オブジェクト表

oid	ref	type	phisical name
oid1	3	dir	"/home/user1"
oid2	1	file	"/home/.world/2"
oid3	1	dir	"/home/.world/3"

図3 世界名前操作表とオブジェクト表の例

(ADD) と、削除 (DEL) がある。例えば図3の最初のエントリは、世界識別子 wid1 の世界において、オブジェクト識別子 oid1 のディレクトリに、ベース名 "file1" が登録され、そのオブジェクト識別子が oid2 であることを意味する。ベース名とは、名前の文字列を'/'を区切り文字としてみた時に、一番右側の文字列である。

オブジェクト表は、オブジェクト (ファイル、または、ディレクトリ) の識別子とその物理名を対応づけるものである。型のフィールド、ガーベジコレクションのための参照カウントのフィールドがある。根世界以外で作られたファイルやディレクトリは、すべてこの表に登録される。ディレクトリの場合、根世界のものも登録されることがある。たとえば、oid1 のディレクトリは、根世界のものである。これは、世界名前操作表におけるＩＤとしての役割と根世界への融合操作の時の対象を保持する役割がある。

オブジェクト表で、物理名が "/home/.world/" から始まっているオブジェクトは、根世界以外の世界で新たに生成されたものである。"/home/.world/" は、3.2で述べたパーティションごとに設けられたディレクトリの名前である。

4.3.1 論理名から物理名への変換

世界 OS サーバは、ファイル名を引数にとるシステムコールを捕捉した時、以下のように論理名と世界識別子から、物理名を求める。まず、ディレクトリ名のオブジェクト識別子を求める。これと、ベース名、与えられた世界識別子（と根世界を除く祖先の世界）で世界名前操作表を検索する。見つかった時には、オブジェクト識別子からオブジェクト表を引いて物理名を求めてそれを返す。見つからなかった時には、Unix のシステムコールを用いて根世界を検索する。根世界で見つかった時には、論理名をそのまま返す。見つからなかった時には、エラーを返す。

4.3.2 世界の操作

世界 OS サーバは、世界の生成の要求を受け付けたとしても、２つの表についてはなにもしない。

世界 OS サーバは、世界の削除の要求を受け付けると、世界名前操作表にある指定された世界識別子を持つエントリを削除する。この時、含まれているオブジェクト識別子 (dir_oid,oid) について、オブジェクト表のエントリの参照カウントを減らす。参照カウントが 0 になった時には、そのエントリを削除する。この時、根世界以外で作られたオブジェクトは、unlink() または rmdir() システムコールを使って実体を削除する。

世界 OS サーバは、世界の融合の要求を受け付けると、２つの表について次の処理を行なう。根世界以外との融合の場合は、世界名前操作表を調べ、子世界の世界識別子を持つエントリの世界識別子を、親世界のものに書き換える。親世界にディレクトリとベース名が同じものがあった場合は、子世界のエントリを残し親世界のものを削除する。根世界との融合の場合は、世界名前操作表を調べ、子世界の世界識別子を持つエントリについて、op_time の順に次の作業を行う。

- 操作が ADD であれば、unlink() または rmdir() システムコールを使って根世界の論理名を削除し、rename() システムコールを使って、子世界のオブジェクトの物理名を論理名へ改名する。
- 操作が DEL であれば、unlink() または rmdir() システムコールを使って、論理名を削除する。

世界 OS サーバは、世界の内容を得る要求を受け付けると、まず、世界名前操作表を調べ、指定された世界識別子を持つエントリ抜きだす。そしてそれらの論理名を表を引いてもとめ、それと操作を項目に持つリストを作る。

4.4 システムコールを捕捉した時の処理

読み込み専用でファイルを開く時や stat() のようにファイルの内容を書き換えないシステムコールを捕捉した時、以下のような処理を行なう。

(1) システムコールの引数を論理名として取り出す。論理名が相対パス名で与えられた時には、プロセスごとに保持しているカレントワーキング・ディレクトリの論理名を使って絶対パス名に変換する。

(2) プロセスが属している世界識別子を取り出す。

(3) 以上の２つを引数として、4.3 節で述べた表を検索する。その結果、物理名と世界識別子、またはエラーが返される。

(4) 物理名が返された場合、システムコールの引数をその物理名に書き換え、システムコールを実行させる。(ただし、論理名と物理名が一致している時には、書き換えない。)

(5) エラーが返された場合、システムコールの実行

を中止し、エラー (ENOENT) を返す。

存在しているファイルを書き込むために開く時や chmod() のように、ファイルの属性を変更するシステムコールを捕捉した時には、(3) で得られた世界識別子と現在のプロセスの世界識別子を比較する。同じ場合は、(4) でシステムコールの引数を物理名に書き換え、システムコールを実行させる。異なる場合、新しく物理名を生成しその名前のファイルを作る。それに (3) で得られた物理名のファイルの内容をコピーし、オブジェクト表に登録する。この時得られたオブジェクト識別子を、世界名前操作表に登録する。システムコールの引数をコピー先のファイルの物理名に書き換えて、システムコールを実行させる。

mkdir() など、新しく名前を登録する必要があるシステムコールを捕捉した時には、(4) でエラー (EEXIST) を返す。(5) で、新しく物理名を作り、オブジェクト表に登録する。この時得られたオブジェクト識別子を、世界名前操作表に登録する。システムコールの引数の論理名を物理名に書き換え、システムコールを実行させる。

getdents() システムコールを捕捉した時は、ファイル記述子からその論理名を得て、4.3節で述べた表を検索し、オブジェクト識別子と物理名を得る。物理名を使って Unix のディレクトリエントリを得る。次に、オブジェクト識別子と世界識別子で世界名前操作表を調べ、これに該当するエントリのベース名を追加、または削除する。最後に、これをシステムコールの結果として返す。

4.5 現　　状

我々は、ここで述べた方法を使って、世界 OS サーバを開発している。現在までに、世界名前操作表とオブジェクト表を操作するモジュールと、4.4節で述べたシステムコールの発行を捕捉した時の処理を実現した。現在、世界の操作を実現している。

本方式に基づきプロトタイプ☆を実現し、次のような環境で性能を測定してみた。

CPU：Ultra SPARC 200Mhz

メモリ：64MB、OS：SunOS 5.5.1

世界の作成の実行時間は 0.9ms、世界の削除、融合、内容を得る操作の実行時間は含んでいるファイルの数に比例し、それぞれファイル 1 つ当たり 11ms、38ms、0.1ms であった。また、世界 OS サーバの管轄下にあるプロセスが発行する open() システムコールの実行時間を測定した。この測定では、ローカルファイルを読み込み専用で開くプログラムを繰り返し実行した。トレースされていない場合、実行時間は 0.07ms、引数の書換えを行わない場合、1.1ms、引数の書換えを行う場合、1.4ms であった。1 章で述べた応用のいくつかは、この性能でも十分実用に耐える。

現在のところ、世界名前操作表とオブジェクト表の永続性を実現していない。また、ディレクトリの属性を変更できないなどの属性に関する問題が残っている。今後は、これらの問題を解決し、1 章で述べた応用プログラムを開発する。

☆ ディレクトリの名前の操作に制限がある。

5. おわりに

この論文では、並列世界モデルに基づくオペレーティング・システムをシステムコールのトレース機能を用いて実現する方法について述べた。並列世界モデルとは、世界とよばれるプロセスの実行環境を複数生成したり、削除したり、融合したりすることを許すモデルである。並列世界モデルを実現する上で重要になる 2 次元の名前空間は、カーネル外のサーバが、システムコールの引数に含まれるファイル名を書き換えることで実現されている。この方法の利点は、カーネルに手を加える必要がない点と、内容の読み書きには一切のオーバーヘッドが掛からないという点にある。サーバの内部では、ファイル名の書き換え先を求める時や、世界の操作を実現するために、世界名前操作表とオブジェクト表の 2 つの表が使われている。

参考文献

1) M.J.Bach : "The Design of the UNIX Operating System", Prentice-Hall (1986).
2) R.Bubenik and W.Zwaenepoel: "Optimistic Make", IEEE Transactions on Computers, Vol.41, No.2, pp.207-217 (1992).
3) B.Ford, M.Hibler, J.Lepreau, P.Tullmann, G.Back, and S.Clawson : "Microkernels Meet Recursive Virtual Machines", Proc. 2nd Symp. on Operating Systems Design and Implementation (OSDI96), pp.137-151 (1996).
4) 當眞, 新城, 根路銘, 溝淵, 喜屋武, 翁長: "投機的処理を支援するオペレーティング・システムにおけるファイル・システム", 情報処理学会研究会報告 95-OS-70-3, Vol.95,No.70, pp.17-24 (1995).
5) 新城、孫: "並列世界モデルに基づくオペレーティング・システム", 情報処理学会コンピュータシステム・シンポジウム, Vol.97, pp.95-100 (1997).
6) Y.Yokote : "The Apertos Reflective Operating System: The Concept and Its Implementation", OOPSLA92, ACM SIGPLAN Notices, Vol.27, No.10, pp.414-434 (1992).