プロセストレース機能を用いた世界OSの実現

石井孝衛[†] 新城靖[††] 板野肯三[††]

世界とは，プロセスの集合とそれらに対する実行環境を合わせたものである．世界は，継承機能と4つの操作（生成，削除，融合，内容一覧）を持つ．世界OSとは，そのような世界を複数提供するOSである．このOSの応用には，ソフトウェアの安全なインストールや破壊からの迅速な回復がある．この論文は，Unix System Vが提供しているプロセストレース機能を用いて世界OSを実現する方法について述べている．この論文で述べてい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とるものに着目し，その引数を書き換えることで世界OSを実現している点にある．カーネルで実現する方法と比較して，この実現方法の優位性は，世界OSの開発と導入が容易になることである．カーネルで実現する方法と異なり，この方法では，プロセストレースによるオーバヘッドが避けられない．この論文では，このオーバヘッドが受け入れ可能な応用があることを示す．

The Implementation of the World OS Using a Process Trace Facility

Kouei Ishii,[†] Yasushi Shinjo[††] and Kozo Itano[††]

A world is integration of processes and their execution environment. Worlds have an inheritance facility and four operations (create, delete, merge, and enumerate contents). The World OS is an operating system that provides multiple worlds. Its applications include safe installation of software and quick recovery from damage. This paper presents the implementation of the World OS by using a process trace facility in Unix System V. One of the main features of the implementation method is to translate the arguments of system calls whose arguments contain file names. Compared with a kernel-based implementation method, this method has the advantage that the development and installation of the World OS become easier. Unlike a kernel-based implementation method, this method involves the inevitable overhead of process tracing. This paper shows that some applications can accept the overhead.

1. はじめに

プログラムの実行によって，ファイルの内容が破壊されることがある．そのような例としては，パッチの適用によるローカル設定の上書き，利用者の誤りによる重要なファイルの削除，また，侵入者の攻撃によるファイルの変更や削除があげられる．そのような破壊からファイルの内容を保護することが重要な課題になっている．

このような破壊からファイルを保護するために，我々は，**並列世界モデル**（**the parallel world model**）に基づくOS（**世界OS**）を開発している．このOSは，世界と呼ばれる抽象を複数提供する．世界とは，プロセスの集合とそれらに対する実行環境を合わせたものである（詳しい定義は，2.1節で示す）．世界の内容のうち重要なものは，ファイルとプロセスである．世界の操作には，**生成**，**削除**，**融合**，**内容一覧の取得**がある．世界には親子関係があり，すべての世界の祖先を**根世界**という．子世界は親世界の内容を継承するが，子世界でのファイルの変更は親世界からは見えない．世界を削除するとその内容も削除される．子世界を親世界と融合すると子世界の内容は親世界に移される．

この論文で述べる世界OSの応用は，次の2つである．

(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ときには子世界を削除する．

(2) 破壊からの迅速な回復：攻撃や利用者の誤りによりファイルが破壊されたときには，その世界を削除することで，迅速に回復させる．

並列世界モデルを実現する方法として，我々は，次の2つの方法を提案してきた．

† 筑波大学大学院理工学研究科
Master's Program in Science and Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

(1) Unix カーネルの名前解決モジュールを変更する方法[9]．名前解決モジュール（`namei()`）とは，`open()` システムコールなどで利用者プロセスから与えられたファイル名をファイルの実体（inode）へマップするモジュールである．このマッピングを，世界ごとに変えることで世界 OS のファイルシステムを実現している．この実現方法の問題点は，カーネルを変更する必要があるので，開発と導入が難しいことである．

(2) 分散型 OS のファイルサーバとして実現する方法[10]．個々のファイルの読み込み手続き，または，書き込み手続きにおいてディスクブロックをアクセスするときに世界ごとに異なる inode を参照することで世界 OS のファイルシステムを実現している．この実現方法の問題点は，このサーバの機能を利用するために既存の応用プログラムを RPC（Remote Procedure Call）を行うように書き換える必要があることである．

このような問題点を解決するために，Unix System V が提供するプロセストレース機能を利用して世界 OS を実現することにした．この論文では，その実現方法について述べる[4]．プロセストレース機能とは，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機能である．この実現方法には，以下のような優位性がある．

- 実現方法 (1) と比較して，カーネルを変更する必要がないので，世界 OS の開発と導入が容易になる．
- 実現方法 (2) と比較して，応用プログラムを変更する必要がないので，世界 OS の利用が容易になる．

カーネルで実現する方法と比較してこの方法の問題点は，プロセストレースによるオーバヘッドが避けられないことである．この論文では，このオーバヘッドが受け入れ可能な応用があることを示す．

この論文は，以下のように構成されている．2 章では，世界の定義，および，世界 OS の機能について述べる．3 章では，世界 OS を利用した安全なソフトウェアの更新を例に用いて世界 OS の概要，および，プロセストレース機能を用いた世界 OS の実現の概要について述べる．4 章では，プロセストレース機能を用いて実現した世界 OS の全体構成について述べる．5 章では，世界 OS のプロセスモジュールについて述べる．プロセスモジュールは，システムコールの捕捉と引数の書き換えを行う．6 章では，世界 OS のファイルモジュールについて述べる．ファイルモジュールは，ファイル名とファイルの実体の，世界ごとのマッピングを実現する．7 章では，プログラムとして，パッチの適用，および，ファイルの削除を用いて，ファイルの保護が実現できたこと，および，プロセストレース機能を用いた世界 OS の実現方式でも十分実用に耐える応用があることを示す．8 章では，関連研究について述べる．

2. 世界の定義と世界 OS の機能

この章では，世界の定義を行い，世界 OS が提供すべき機能について述べる．

2.1 世界の定義

世界とは，プロセスの集合とそれらに対する実行環境を合わせたものである．世界の内容のうち重要なものは，ファイルとプロセスである．世界を抽象データ型としてとらえ，次のような操作があるものとして規定する．

生成 既存の世界を継承して，新たな世界を生成する．新しく生成された世界を子世界（a child world），元になった世界を親世界（a parent world）と呼ぶ．継承とは，子世界で上書き（override）された内容を除き，親世界の内容を参照可能にすることである．子世界は，親世界との差分を持つ．こうして定義される親子関係により，システムの中の世界は，**DAG**（**Directed Acyclic Graph**）を構成する（**図 1**）．

削除 世界の内容であるプロセスやファイルを削除する．さらに，その世界に依存している世界（子孫

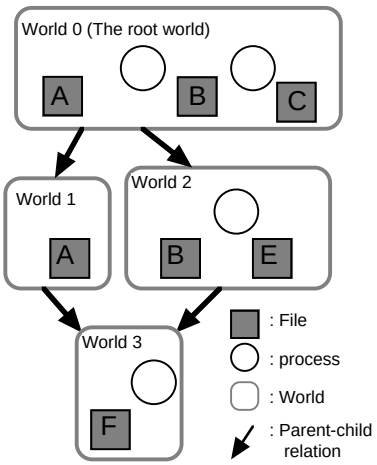

図 1 世界の DAG
Fig. 1 A DAG of worlds.

の世界）も再帰的に削除する．

融合 融合する世界が持つ内容を融合される世界に移し，融合する世界を削除する．同一の名前のファイルがあれば，融合する世界のものを残し，融合される世界のものを削除する．融合する世界に子孫の世界があれば，それを融合される世界の子孫とする．

内容一覧 世界の内容であるファイルとプロセスのリストを返す．

世界という考え方は，投機的処理（speculative processing）（または，楽観的処理（optimistic processing））を，オペレーティングシステムのレベルで支援するために考案された．ここで，投機的処理とは，それが将来利用されるか利用されないか確定される前に実行を開始する処理である．投機的処理の目的は，みかけの処理時間の短縮である．たとえば，投機的 make[9] は，ソースファイルの変更を検出すると，利用者が make と打つ前に，子世界を生成してコンパイルなどの処理を開始する．利用者が make と打つと，投機的 make は，生成した世界を必須の世界に融合する．このとき，コンパイル作業が完了していれば，利用者には瞬時にコンパイル作業が終了したように見える．利用者が make と打つ前に再びソースファイルを変更した場合，投機的 make は，生成した世界を削除する．

投機的処理を支援するうえで重要なことは，投機的処理を実行するために作成した子世界におけるプロセスが，融合するまで必須の世界である親世界にけっして影響を与えてはならないということである．そのため異なる世界に属するプロセス間の通信を，それらの世界が融合されるまで遅延させる機能が必要である．この点について詳しくは，3.3 節で議論する．

世界 OS は，世界の操作を通じて一種の隔離された実行環境を提供する．このような実行環境を提供するシステムの例としては，仮想計算機に基づくシステムやマイクロカーネルに基づくシステムがある．これらのシステムと比較して，世界 OS の特徴は，世界の中の実行環境が，継承や融合により他の世界の中の実行環境と強く結び付いている点にある．

世界 OS は，世界の操作を通じてプロセスとファイルをグループ化する機能を提供する．たとえば，世界 OS では，世界の削除により同一の世界に属するプロセスとファイルを一度に削除することができる．これに対して Unix では，シグナルの送信においてプロセスをまとめて扱う機能がある．この機能は，ジョブ制御やログオフ時のプロセスの削除に使われている．Unix におけるグループ化の機能と比較して，世界 OS におけるグループ化の機能の特徴は，1 つのグループにプロセスやファイルという異なった種類のオブジェクトを含めることができること，グループ間に親子関係があること，および，グループ間に演算（融合）が定義されていることにある．

表 1 世界 OS 特有の機能
Table 1 The functions particular to World OS.

機能	システムコール	コマンド
世界の生成	`world_create()`	`wcreate`
世界の削除	`world_delete()`	`wdelete`
世界の融合	`world_merge()`	`wmerge`
世界の内容一覧	`world_contents()`	`wcontents`
プロセスの操作	`wfork()` `process_create()` `chworld()`	`wexec`

2.2 世界 OS が提供する機能

世界 OS は，通常のオペレーティングシステムが提供する機能（ファイルの作成，読み書き，プロセスの生成，プログラム実行など）に加えて，世界 OS 独自の機能を提供する．世界 OS が提供する独自の機能を，**表 1** に示す．表 1 において最初の 4 つは，それぞれ 2.1 節で述べた世界に対する 4 つの操作と対応している．そのほかに，異なる世界でプログラムを実行するための機能がある．コマンドレベルでは，wexec コマンドを提供している．システムコールレベルでは，次の 3 つの方法を提供している．

`wfork()` Unix の `fork()` システムコールと同様に，コピーによりプロセスを生成する．ただし，生成されたプロセスは引数で指定された世界で動作する[6]．

`process_create()` コピーではなく，プログラムを指定してプロセスを生成する．`wfork()` と Unix の `execve()` を合わせたようなもの[6]．

`chworld()` プロセスが属している世界を，引数で指定されたものに変更する．

`chworld()` は，今回新たに導入したシステムコールである．`chworld()` の使用によって投機的処理のセマンティクスが保てなくなる場合があるため，過去の世界 OS[9),10)] では `chworld()` を提供していない．この点について詳しくは，3.3 節で議論する．

3. 世界 OS の利用例とプロセストレース機能を用いた実現の概要

この章では，安全なソフトウェアの更新を例として世界 OS の概要について述べる．また，プロセストレース機能を利用した実現方法の概要について述べる．

3.1 世界 OS を利用した安全なソフトウェアの更新

世界 OS を利用してソフトウェアの更新を行うことで，次のようなことが可能となる．

(1) たとえ競合するファイルがあったとしても，元の状態を保持したまま，新しいファイルをインストールできる．
(2) 作業中に生成，更新，削除されるファイルの一覧を得ることができる．
(3) 実験的にプログラムを実行して，その動作を確認することができる．
(4) 問題が発生したときに元に戻すことができる．

この節では，World Wide Web（以下 WWW）サーバ Apache を更新する例を用いて，世界 OS の利用について述べる．

世界 OS において，ソフトウェアの安全なインストールを行うには，システム管理者は，まず新たに作業用の世界を生成する．シェルのレベルで世界の生成を支援するために，世界 OS は，`wcreate` コマンドを提供している．`wcreate` コマンドの利用方法を以下に示す．

```
# wcreate 0
world 120 created
# wexec 120 tcsh
# make install
cp httpd.conf /etc/httpd.conf
cp httpd /usr/sbin/httpd
# apachectl start
apachectl start: httpd started
# _
```

`wcreate` は，世界の生成を行うコマンドである．このコマンドは，引数として，内容を継承する世界（親世界）の世界の識別子（以下 **WID**）を 1 つ以上とる．この例では，根世界（WID=0）を継承して，新たに世界を生成している．生成された世界は，根世界の子世界になり，その WID は 120 である．`wexec` は，世界を指定してプログラムの実行を行うコマンドである．この例では，WID が 120 の世界で，シェル `tcsh` を実行している．そして更新を行うコマンド（`make`）を実行し，WWW サーバ `httpd` を起動している．

インストール作業が完了したとき，管理者は，作業の結果どのようなファイルが更新されたか，および，どのようなプロセスが生成されたかを知りたくなる．その理由は，作業のときに実行されたスクリプトによりローカルの設定が上書きされたり，意図していないプロセスが残されている可能性があるからである．そこで，世界 OS では，作業の結果，どのようなファイルが更新・追加されたか，および，どのようなプロセスが残されたかを調べるためのコマンド `wcontents` を提供している．

```
# wcontents 120
files:
+ Mon Oct 10 21:07:20 2001  /etc/httpd.conf
+ Mon Oct 10 21:07:19 2001  /usr/sbin/httpd
...
processes:
1056 /usr/sbin/httpd
1057 /usr/sbin/httpd
...
# _
```

この例では，子世界（WID=120）で 2 つのファイル `"/etc/httpd.conf"` と `"/usr/sbin/httpd"` が更新されている．この実行例では示されていないが，親世界（根世界）では更新される前のファイルが保持されている．このように管理者は，`wcontents` コマンドによって，更新されたファイル名を調べたり，更新されたプログラムを実際に動かしてみることで，更新作業による影響を調べることができる．

更新作業の結果，問題が見つかった場合，管理者は，以下のように `wdelete` コマンドを実行して子世界を削除する．

```
# wdelete 120
world 120 was deleted.
# _
```

このコマンドの実行によって，子世界（WID=120）で更新されたファイルやプロセスも削除される．

一方，問題が見つからなかった場合，管理者は，以下のように `wmerge` コマンドを実行して子世界と根世界を融合することができる．

```
# wmerge 0 120
world 0 and 120 were merged
# _
```

このコマンドの実行によって，子世界（WID=120）で更新・追加されたファイルは親世界（WID=0）に反映される．また，子世界で生成されたプロセスは親世界に移される．

3.2 論理名と物理名という概念を用いた世界 OS のファイルシステム

世界 OS では同じ名前のファイルでも世界が違えば異なる実体を指すようにしなければならない．これを実現する方法として，従来の世界 OS の実現では，1 章で述べたように，inode を用いていた．しかしながら，プロセストレース機能を用いてカーネル外のサー

バプロセスで世界 OS を実現する場合，inode を用いることはできない．

この論文では，カーネル外のサーバプロセスで世界 OS を実現するために，次の 2 種類のファイル名を用いる方法を提案する．

論理名：ある世界でファイルを一意に特定できるファイル名．

物理名：ファイルの実体と 1 対 1 に対応したファイル名．

利用者プロセスは，論理名だけを扱うことができる．論理名だけではファイルの実体を特定できないが，利用者プロセスの所属する世界の WID を加えることでファイルの実体，すなわち物理名を一意に特定できるようになる．同じ論理名が与えられた場合でも，WID に応じて別の物理名を与えることで，世界 OS のファイルシステムが実現される．

3.1節で述べた例では,子世界で `"/etc/httpd.conf"` と `"/usr/sbin/httpd"` の 2 つのファイルが更新された．その際，次のようなシステムコールが実行されている．

```
open("/etc/httpd.conf",O_WRONLY)
open("/usr/sbin/httpd",O_WRONLY)
```

この場合，このシステムコールを実行前に停止させ，新しい物理名を生成し，元のファイルの内容をコピーする（copy-on-write）．物理名を，パーティションごとに設けた隠しディレクトリを基点として格納する．この隠しディレクトリを **dot-world ディレクトリ**と呼ぶ．ここでは，`"/.world/101"` と `"/.world/102"` という物理名を生成したものとする．そしてシステムコールの引数を次のように書き換えて実行を再開させる．

```
open("/.world/101",O_WRONLY)
open("/.world/102",O_WRONLY)
```

システムコールの実行直後には，もう一度停止させ，書き換えた引数を元に戻す．

世界の操作には，生成，削除，融合，内容一覧の 4 つがある．

世界の生成では，新たにユニークな WID を生成する．

世界の削除では，指定された WID に関連付けられたファイルを削除する．3.1 節で述べた例では，世界の削除の際に次のようなシステムコールを実行する．

```
unlink("/.world/101")
unlink("/.world/102")
```

世界の融合では，指定された子世界の内容（ファイルとプロセス）を，別の指定された親世界に移動する．3.1 節の例では次のようなシステムコールを実行する．

```
rename("/.world/101","/etc/httpd.conf")
rename("/.world/102","/usr/sbin/httpd")
```

世界の内容一覧では，指定された WID に関連付けられたファイルの一覧を生成する．そのために内部の表（永続性がある）を用いてファイルを特定する．ファイルは論理名に変換して，利用者プロセスに渡す．

3.3 プロセスの扱い

プロセストレース機能を用いた世界 OS の実現では，アプリケーションのプロセスについては主に所属する世界の WID を内部的な表で管理するだけでよく，ファイルシステムのようにシステムコールの引数を書き換える処理は必要ない．ただし，世界の削除では，削除対象の WID を持つプロセスを削除する．3.1 節の例では，`wdelete` コマンドが実行されたとき，次のようなシステムコールを実行する．

```
kill(1056,SIGKILL)
kill(1057,SIGKILL)
```

異なる世界にプロセスを生成する方法として，**chworld()** というシステムコールを導入する．`chworld()` は，プロセス自身が所属する世界を変更するシステムコールである．3.1 節で述べた `wexec` コマンドの実行では，まずシェルが `fork()` システムコールと `execve()` システムコールを実行して `wexec` コマンドのプロセスを生成している．次に `wexec` コマンドのプロセスが `chworld()` システムコールを実行した後に `execve()` システムコールを実行している．この結果，異なる世界にプロセスが生成される．ただし，この方法では，ファイルを開いたままで `chworld()` システムコールを実行すると，元の世界のファイルが移動した先の世界でも見えてしまう．すなわち，`chworld()` システムコールを実行する以前に有効になっているファイル記述子は，`chworld()` 後も引き続き有効である．このことは，投機的処理では，後で利用されないことが確定して削除すべきプロセスによる活動の結果が，必須の世界に影響を与えてしまうことを意味し，従来の世界 OS の応用では問題になる．

一方，1 章で述べた世界 OS の応用（ソフトウェアの安全なインストール，および，破壊からの迅速な回復）では，積極的に子世界の内容を利用者に提示するものであり，投機的処理ほどの厳密性は要求されていない．さらに，プロセストレース機能を用いた実現では，4 章で述べるように捕捉するシステムコールの数を減らすことが重要になるが，開いたファイルを渡すことを禁止するためには，ファイル記述子を引数にとるようなシステムコールを捕捉する必要が生じる．よって今

回は，chworld() システムコールにおいて開いたファイルをそのまま渡す仕様にした．

プロセス間通信については，投機的処理を支援するためには，融合されるまで遅延させる機能が必要である[9]．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機能が必要である．一方，1 章で述べた世界 OS の応用（ソフトウェアの安全なインストール，および，破壊からの迅速な回復）では，子世界においてシェルを実行しその結果を利用者に提示する必要がある．このとき，異なる世界にあるプロセスが，ウィンドウシステムや疑似端末を通じて通信する必要がある．よって今回の実現では，融合されるまで遅延させる機能を設けず，プロセス間通信をそのまま実行させる仕様にした．

3.4 世界の融合のセマンティクス

世界 OS では，融合のセマンティクスとしては，子世界優先と最終更新時刻優先の 2 種類がある．プロセストレース機能を用いた世界 OS の実現では，今まで開発してきた他の世界 OS と同様に，子世界優先セマンティクスを用いる．なお，世界 OS が対象とする応用（文献 9) で議論した投機的処理，および，1 章で述べたソフトウェアの安全なインストールと破壊からの迅速な回復）では，厳密な直列化可能性が求められていないので，直列化可能ではないような融合を許している．たとえば，ソフトウェアの更新では，子世界で書き換えられたファイルと同じ論理名を持つファイルが融合の前に別の子世界で読み込まれた場合に，融合を許したとしても，何の問題も発生しない．

3.5 世界のアクセス制御

世界について次のようなアクセス制御を導入する．

- 利用者 ID（以下 UID）に基づくアクセス制御：Unix におけるプロセスに対するアクセス制御と同様に，同じ UID を持つプロセスに対して削除と融合を許す．生成では，異なる UID を持つ世界を親世界として指定することを許す．
- 所属する世界に基づくアクセス制御：プロセスが所属する世界の子孫の世界に対する世界の操作のみを許す．

4. 世界 OS の全体構成

3 章では，世界 OS の概要と，プロセストレース機能を用いた実現方法の概要について述べた．この章では，世界 OS の全体構成について述べる．

プロセストレース機能を用いた世界 OS の構成を，

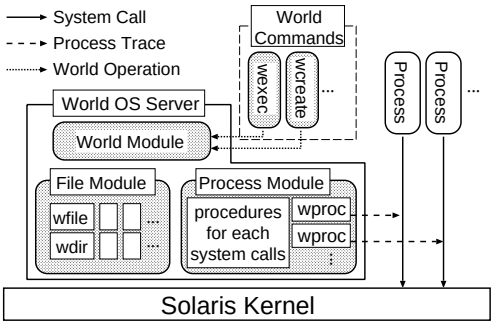

図 2 プロセストレース機能を用いた世界 OS の構成
Fig. 2 Organization of the World OS using a process trace facility.

図 2 に示す．世界 OS は，変更を加えていない Solaris カーネルと，その上で動作する世界 OS サーバから構成される．世界 OS サーバは，プロセスモジュール，ファイルモジュール，世界モジュールから構成される．

プロセスモジュールは，プロセストレース機能を利用して，通常の Unix のシステムコールの一部を捕捉し，その引数に現れるファイル名や結果を書き換える．プロセスモジュールの詳細については，5 章で述べる．

世界モジュールは，世界の親子関係を管理する．世界モジュールは，世界の親子関係を独自の形式でファイルに保存している．世界 OS サーバ内部に対して，get_ancestors という手続きを提供する．get_ancestors は，指定された世界の，すべての祖先の世界のリストを返す[10]．このリストは，指定された世界との親子関係の近い順に整列される．また，世界モジュールは，Sun RPC[11] を用いて，2.2 節や 3.1 節で述べたコマンド群からの要求を受け付ける機能を持つ．

ファイルモジュールは，論理名と物理名の関連付けを管理する．ファイルモジュールで最も重要な手続きは，wdir_lookup である．これは，入力として，論理名，カレントワーキングディレクトリ，ルートディレクトリ，WID，利用者権限をとる．そして，出力として，物理名の有無と，物理名，見つかった世界の WID を返す．ファイルモジュールの実現については，6 章で述べる．

この論文で述べる世界 OS の実現方式は，単一のホストで使用できるものである．NFS などを用いてファイルを共有している複数のホストで動作するような，ネットワーク型の世界 OS を実現するには，文献 10) で示したようなネットワーク型の世界サーバが必要になる．ただし，ホストごとに異なる dot-world ディレクトリを用いることで，NFS でマウントしたファイ

ルシステムにおいても，本実現方式で世界 OS を実現することができる．

5. プロセスモジュール

プロセスモジュールは，プロセストレース機能を利用して，通常の Unix のシステムコールの一部を捕捉し，その引数に現れるファイル名や結果を書き換える．プロセスモジュールは，内部に，捕捉するシステムコールごとの手続きを持つ．また，プロセストレース機能を利用するために，プロセスごとに **wproc** クラスのインスタンスを持つ．この章では，これらの実現について述べる．

5.1 システムコールの捕捉

プロセスモジュールが捕捉するシステムコールを，**表 2** に示す．本方式の特徴は，高速化のために捕捉するシステムコールを減らしている点にある．まず，本システムは `read()` システムコールや `write()` システムコールを捕捉しない．これにより，ファイルの内容を読み書きする処理を，オーバヘッドなしに実行させることができる．次に，ファイルの属性を扱うシステムコールのうち `fchmod()` や `fchown()`，`fstat()` などを無視する．その代わりに，属性の変更についても，内容の書き込みと同様にファイル全体をコピーしている．これによりファイルのコピーによるオーバヘッドが生まれるが，多くの応用では属性の変更を行うことはまれであるため，平均的には，システムコールを無視することによる利益がコピーのオーバヘッドを上回る．たとえば，7 章の実験で用いたパッチの適用では，これらのシステムコールを無視するようにしたところ，システムコールの捕捉回数が約 23%減少し，実行時間が 8.6%短縮した．

5.2 システムコールを捕捉したときの手続き

システムコールを捕捉したときの手続きを，代表的な例として `stat()` と `open()` を用いて説明する．

表 2 捕捉するシステムコール
Table 2 System calls to be traced.

(1)読み込み	open(RDONLY), open64(RDONLY), exec(), execve(), access(), readlink(), sysfs(), acct(), pathconf(), statfs(), statvfs(), stat(), lstat(), stat64(), lstat64(), acl(GETACL), chdir(), fchdir(), chroot(), fchroot(), resolvepath()
(2)生成、更新	open(), open64(), create(), create64(), chmod(), chown(), utime(), utimes(), lchown(), acl(SETACL)
(3)登録、削除	link(), unlink(), mkdir(), rmdir(), mknod(), symlink(), rename()
(4)getdents()	getdents(), getdents64()
(5)fd関連	dup()
(6)fork()	fork(), fork1(), vfork()

`stat()` は，ファイルに対する更新のないシステムコールであり，`open()` は，ファイルを読むほか更新や生成を行うシステムコールである．

`stat()` を捕捉したときは，まず，`stat()` の引数に現れたファイル名（論理名）と，プロセスのカレントワーキングディレクトリ（以下 cwd），ルートディレクトリ，WID，権限を得る．ここでいう権限とは，プロセスの UID や GID などのことである．これらを引数として，`wdir_lookup` を呼ぶ．その結果，5 通りの結果が得られ，それぞれについて以下のような処理を行う．

- 同じ世界で物理名が見つかった：引数に現れた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．
- 根以外の祖先の世界で物理名が見つかった：同じ世界で見つかったときと同じ処理を行う．
- 根世界で物理名が見つかった：引数を書き換えずにシステムコールを実行させる．
- 物理名が見つからなかった：システムコールを中止し，システムコールの返戻値を -1 に，エラー番号を `ENOENT` に変更する．
- エラー：システムコールを中止し，返戻値とエラー番号を，エラーに対応する値に変更する．

システムコールの実行後には，書き換えた引数を元に戻す．

`open()` を捕捉したときは，`open()` の第 2 引数を得る．これが `O_RDONLY` であれば，`stat()` と同じ処理を行う．そうでなければ，`stat()` の処理と同様に `wdir_lookup` を呼ぶ．その結果，5 通りの結果が得られ，それぞれについて以下のような処理を行う．

- 同じ世界で物理名が見つかった：引数に現れた論理名を物理名に書き換える．
- 根以外の祖先の世界で物理名が見つかった：新しく物理名を生成し，その名前を持つファイルを作る．祖先の世界で見つかった物理名を持つファイルの内容を，新しく生成したファイルにコピーする（copy-on-write）．引数に現れた論理名を，新しく生成した物理名に書き換え，システムコールを実行させる．また，ファイルモジュールに，WID で示された世界で論理名が追加されたことを登録する．
- 根世界で物理名が見つかった：根以外の祖先の世界で物理名が見つかったときと同じ処理をする．
- 物理名が見つからなかった：`open` の第 2 引数を調べる．これに `O_CREAT` が含まれていれば，新しく物理名を生成し，引数をこの新しい物理名に書

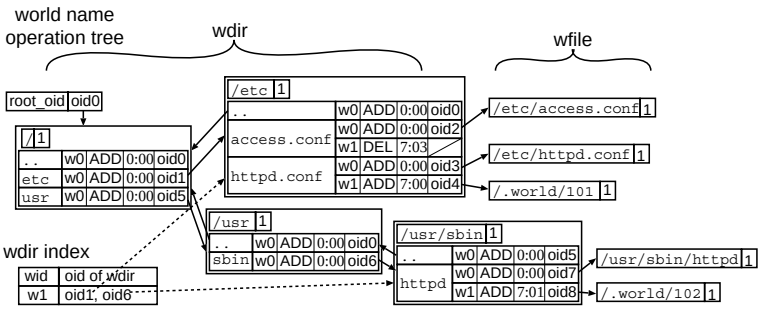

図 3 世界名前操作木と wdir 索引
Fig. 3 The World name operation tree and the wdir index.

き換えてシステムコールを実行させる．O_CREAT が含まれていなければ，システムコールを中止し，システムコールの返戻値を −1 に，エラー番号を ENOENT に変更する．

- エラー：システムコールを中止し，返戻値とエラー番号を，エラーに対応する値に変更する．

システムコールの実行後に，書き換えた引数を元に戻す．また，システムコールの返戻値から，ファイル記述子を得て，ファイル記述子と開いたファイルとの対応を記録する．この対応は，getdents() などのファイル記述子を引数に持つシステムコールを捕捉したときに使用する．

5.3 wproc クラス

wproc クラスは，システムコールの引数や結果の書き換えを実現するためのクラスである．本システムは，プロセスごとに，wproc クラスのインスタンスを生成する．wproc の属性には，プロセス ID（以下 PID），所属する世界の WID，exec() または execve() システムコールを実行したときのプログラム名，プロセスの cwd やルートディレクトリ，および，ファイル記述子と開いているファイルとの関連を記録する表などがある．

5.4 世界の操作におけるプロセスへの処理

世界の操作（2.1 節で述べた，生成，削除，融合，および，内容一覧）の際には，同じ世界に属するプロセスをまとめて操作する必要がある．そのためにプロセスが所属する世界を記録する表を持つ．この表の各エントリは，WID と，wproc オブジェクトのリストである．

世界を生成するときは，指定された WID を持つエントリを作る．世界を削除するときは，指定された WID を持つエントリを削除し，またそこにある wproc の示すプロセスを kill() によって削除する．世界を融合するときには，指定された子世界の WID を持つエントリを，指定された親世界の WID を持つエントリに移す．内容の一覧を求められたときには，指定された世界の WID を持つエントリを調べ，wproc の PID とプログラム名を返す．

6. ファイルモジュール

ファイルモジュールの主要な役割は，ファイルの論理名と物理名の対応関係を管理することである．最も重要な手続きは，wdir_lookup である．ファイルモジュールの実現の特徴は，世界名前操作木と呼ばれるデータ構造を用いている点にある．また，ファイルについての世界の操作を実現するために，wdir 索引と呼ばれるデータ構造を含む．

6.1 世界名前操作木と wdir 索引

世界名前操作木は，ファイル名への操作（追加や削除）を記録する木である（**図 3**）．世界名前操作木の中間節を **wdir** オブジェクト，端節を **wfile** オブジェクトと呼ぶ．wfile と wdir は固定長のオブジェクト識別子（以下 **OID**）により識別子される．wfile は，物理名と参照数を持つ．wdir は，wfile の要素に加えて，ディレクトリエントリのリストを持つ．このディレクトリエントリは，ある世界でのファイルへの操作を記録するものである．各ディレクトリエントリは，ベース名（ファイル名のうち「/」で区切られた文字列），WID，操作，操作が行われた時刻，OID を持つ．操作には，**ADD** と **DEL** がある．操作が ADD のディレクトリエントリは，WID が示す世界で名前が追加されたことを示す．操作が DEL であるディレクトリエントリは，WID が示す世界で名前が削除されたことを示す．たとえば，図 3 の，物理名が "/etc" である wdir は，ベース名が "httpd.conf"，WID が w1，操作が ADD のディレクトリエントリを持つ．これは，世界 w1 で名前 "/etc/httpd.conf" が追加されたことを示している．また，ベース名が "access.conf"，WID が w1，操作が DEL のディレクトリエントリは，世界 w1 で名前 "/etc/access.conf" が削除された

ことを示している．

wdir 索引は，根以外の世界の WID を含むディレクトリエントリを持つ wdir オブジェクトを，その世界ごとにまとめた表である．wdir 索引の各エントリは，WID と，wdir の OID のリストである．

ファイルモジュールは，世界名前操作木に加えて，Unix のディレクトリ木も利用して，論理名と物理名の対応関係を管理する．Unix のディレクトリ木の役割は，ファイルの実体を保持することである．ファイルモジュールは，根以外の世界のファイルの実体を，dot-world ディレクトリに，数字を用いて一意な名前を生成して記録する．dot-world ディレクトリは，各パーティションごとに用意する．根世界に属するファイルの実体は，Unix のディレクトリ木に論理名を使って直接記録する．

世界名前操作木には，根世界の WID を持つエントリもキャッシュとして追加する．世界 OS サーバが一番最初に起動するとき，世界名前操作木に物理名として "/" を持つ wdir を登録する．それ以外の節やディレクトリエントリを，世界名前操作木の検索の過程で追加する．たとえば図 3 では，世界 w1 で "/etc/httpd.conf" という論理名の有無を調べる際に，物理名として "/etc" を持つ wdir を追加している．

wfile オブジェクト，wdir オブジェクト，および，wdir 索引は，B+木を用いてファイルに保存している．このとき，キーとして wfile と wdir は OID，wdir 索引は WID を用いている．

ファイルモジュールは，世界名前操作木を検索する手続きとして，`wdir_lookup()` を提供する．この手続きには，通常の Unix の名前解決モジュールが用いているアルゴリズムと似たアルゴリズムを用いている．違いは，ファイル名に加え WID（祖先の世界のものも含む）を用いている点，および，根世界の WID を持つエントリを，キャッシュとして世界名前操作木に登録している点にある．詳しいアルゴリズムを付録 A.2 に示す．

6.2 ファイルモジュールにおける世界の操作

世界の生成が要求されたとき，ファイルモジュールは，引数で指定された世界の WID を持つエントリを wdir 索引に追加する．

世界の削除が要求されたとき，ファイルモジュールは，指定された世界で行われたファイル操作の記録をすべて削除しなければならない．この操作では，まず，wdir 索引から指定された WID を持つエントリを得る．そのエントリに含まれている wdir の OID のリストから，**図 4** のようなファイル名への操作のリストを作る．図 4 に示した操作リストは，図 3 において世界 w1 における名前の操作を集めたものである．このリストの各要素は，その操作が行われた wdir の OID，ベース名，操作，時刻，および，OID（操作が ADD のときのみ）から構成される．このリストを時刻の順にソートする．世界の削除では，時刻が新しい順に，このリストに載っているディレクトリエントリを削除する．すなわち，その wdir オブジェクトにおいて，ベース名と WID をキーとしてディレクトリエントリを検索し，見つかったディレクトリエントリを削除する．このとき，ディレクトリエントリの操作が ADD であれば，それが示すオブジェクトの参照数を減らす．これにより参照数が 0 になった場合，そのオブジェクトを削除し，その物理名を `unlink()` または `rmdir()` によって削除する．操作が DEL であれば，単にディレクトリエントリを削除する．

oid of wdir	base name	op	op time	oid
oid1	`httpd.conf`	ADD	7:00	oid4
oid6	`httpd`	ADD	7:01	oid8
oid1	`httpd.conf`	DEL	7:03	

図 4 世界 w1 での操作リスト
Fig. 4 The operation list of the World w1.

世界の融合が要求されたとき，ファイルモジュールは，指定された世界で行われたファイル操作を，別の指定された世界に移さなければならない．この場合，世界の削除と同様に操作リストを作成する．そして，子世界の WID を親世界のものに変更する．ただし，融合先が根世界である場合には，wfile や wdir の論理名を求め，`rename()` システムコールを使って物理名を論理名と同じものに変更する．

世界の内容一覧が要求されたとき，ファイルモジュールは，指定された世界に属するファイルの名前と操作の一覧を返さなければならない．この操作の入力は，WID，一覧を格納するバッファ，および，そのサイズである．この場合も，世界の削除や融合と同様に，まず操作リストを作る．次に，操作リストから論理名を作り，それと操作の組を作る．論理名を求める手続きは，ライブラリ関数 `getcwd()` と同じように，親ディレクトリの論理名を再帰的に調べて，その論理名の後ろにベース名を結合する．

7. 実　　験

カーネルレベルの実現方法と比較して，プロセストレース機能を用いた実現方法では，プロセストレースやファイル単位の copy-on-write のオーバヘッドがある．そのオーバヘッドがアプリケーションの実行時間

表 3 プログラムと世界の操作の実行時間
Table 3 Execution times of programs and world operations.

programs	number of traced system calls	files that the program created, modified or deleted	the execution time of the programs		the execution times of world operations			
			Solaris 8	World OS	create	delete	merge	contents
patch	15557	16（3.6 MB）	54 s	71 s（+31%）	0.39 ms	0.20 s	0.94 s	0.012 s
rm	827	366（23.1 MB）	4.3 s	0.69 s（−84%）	0.39 ms	2.1 ms	4.4 s	0.027 s

に及ぼす影響を調べるために実験を行った．本章ではその結果を示す．

ファイルの更新をともなう処理として，以下の 2 つのプログラムを用いて実験を行った．

- Solaris8 のパッチの適用（パッケージ管理プログラムの修正．パッチ番号 110934-01）
- rm コマンドの実行（本システムのソースプログラムとオブジェクトコードを納めたディレクトリの削除）

この 2 つのプログラムは，1 章で述べた世界 OS の応用の典型的な例になっている．前者は，ソフトウェアの安全なインストールの例である．後者は，破壊からの迅速な回復の例である．これらのプログラムを，通常の Solaris と本システムの上で実行した．

実験環境は，次のとおりである．世界 OS サーバを実行させるマシンには，Sun Blade 1000 を用いた．そのオペレーティングシステムは Solaris8，その CPU は UltraSPARCIII 750 MHz，メモリは 512 MB，2 次記憶はファイバチャネル接続，容量 18.2 GB で，データ転送速度 100 MB/s である．

この実験の結果を**表 3** に示す．世界 OS 上でプログラムを実行した場合，Solaris と比較して，パッチの適用では 31%のオーバヘッドがあった．この程度のオーバヘッドでファイルを保護することができるため，本システムは有用であると考える．一方 rm コマンドの実験では，通常の Solaris と比べて短時間で完了した．これは，unlink() システムコールや rmdir() システムコールの実行を世界の融合まで遅延したことによる．

世界の操作のうち，生成の実行時間は，0.39 ms と非常に速い．これは，内部的な表のエントリを追加するだけで実現可能であるからである．

世界の削除，および，融合の実行時間は，アプリケーションの振舞いに依存する．パッチの適用では，15 個のファイルを生成または更新し，1 個のディレクトリを生成している．そのため，世界の削除では 15 回の unlink() システムコールと 1 回の rmdir() システムコールを実行する．実行時間は 0.20 秒であった．世界の融合では，16 回の rename() システムコールを実行する．実行時間は 0.94 秒であった．rm コマンドの実行では，360 個のファイルと 6 個のディレクトリを削除している．rm コマンドの実行では，生成または更新したファイルがないため，世界の削除では内部のデータ構造の変更だけを行う．そのため，パッチの適用と比べてファイルの数が多いにもかかわらずパッチの適用よりも短時間で削除が完了した．一方，世界の融合では，360 回の unlink() システムコールと 6 回の rmdir() システムコールを実行する．これは rm コマンドを直接実行したことと等価であり，その実行時間は，Solaris 上で rm コマンドを実行したときの時間と同等であった．

8. 関連研究

8.1 システムコールのトレースを行う研究

Ufo[2] は，ftp や http などのプロトコルを通じて提供されるインターネット上の資源を，ローカルのファイルと同じように参照できるようにするためのファイルシステムである．このシステムでは，プロセストレース機能を用いて，open() などのシステムコールの引数を，ダウンロードしたファイルの名前に書き換えている．

システムコールを捕捉してアクセス制御を強化する研究としては，Planet[12]，MAPbox[1]，細粒度保護ドメインを実現するカーネル[8] などがある．これらのシステムでは，実行直前にシステムコールを停止しその引数を調べ，不当なものはその引数を書き換えたりシステムコールの実行を中止させたりすることで，アクセス制御を強化する．

これらの研究と比較して本研究の特徴は，ファイルの copy-on-write を行うことでファイルの内容を保護している点，および，世界という単位でファイルの内容の回復を容易に行える点である．

8.2 ファイルの更新を隔離する研究

投機的処理を支援する V System[3] の研究では，encapsulation と呼ばれる仕組みを V System に導入している．encapsulation は，本研究の世界と同様に，生成，削除，および，融合に相当する必須化という機能がある．encapsule されたファイルは，encapsulation

ごとに作られたディレクトリ木に保存される．encapsulationと比較して本研究の特徴は，柔軟性と実現方法にある．encapsulationでは，必須化（根世界への融合に相当）しかできず，また直列化可能でない場合は，必須化が失敗する．並列世界モデルでは，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．また，この論文で述べた実現方式では，プロセストレース機能を用いている点，および，世界ごとのディレクトリ木を作らず，6章で述べた世界名前操作木を用いている点が異なる．

Elephant file system[7]は，ブロック単位でcopy-on-writeを行うことで，ファイルの内容の更新履歴を記録する．履歴は利用者が指定する指針に従って自動的に破棄される．このシステムは，FreeBSDのカーネルを変更して実現されている．このシステムと比較して本システムの特徴は，プロセストレース機能を利用してカーネル外で実現している点，および，世界の融合がある点である．

Sun OSのTFS（Translucent File System）[11]と，BSDのunionファイルシステム[5]は，あるディレクトリ木の上に別のディレクトリ木を重ね合わせる機能を持つ．これらのファイルシステムでは，重ね合わせたディレクトリ木のすべての内容が統合された状態を参照することができる．ただし，書き込みは，一番上に重ねられたディレクトリ木に行われる．これらのファイルシステムと比較して本研究の特徴は，世界がファイルに加えプロセスを含んでいる点，および，変更結果を元のファイルに反映させる手続き（世界の融合）がある点である．

9. おわりに

この論文では，Unix System Vが提供しているプロセストレース機能を用いて世界OSを実現する方法について述べた．世界OSとは，世界と呼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OSである．このOSの応用には，ソフトウェアの安全なインストールや破壊からの迅速な回復がある．

この論文で述べた世界OSの実現方法の特徴は，第1に，システムコールの引数としてファイルの名前をとるものに着目し，その引数を書き換える点にある．従来の実現方法と比較してこの実現方法の優位性は，カーネルまたは応用プログラムを変更する必要がないので，世界OSの開発と導入が容易になることである．第2の特徴は，ファイル名に対する操作を世界名前操作木とwdir索引を用いて記録している点にある．実験により，パッチを当てる作業では，31%のオーバヘッドにより世界により保護された環境が利用可能になることが分かった．このことから，プロセストレース機能を用いた世界OSの実現方式でも，十分実用に耐える応用があるといえる．また，世界の生成は高速に実行可能であり，その実行時間は0.39ミリ秒であった．世界の削除と融合の実行時間は，アプリケーションの振舞いに大きく依存する．たとえば，ファイルの数が16個程度のとき，世界の削除は0.20秒，融合は0.94秒であった．

今後の課題は，複数の世界に属するファイルを操作する機能を実現することである．この機能を用いることで，たとえば親と子の世界にある同名のファイルの内容を比較することが可能になる．

参考文献

1) Acharya, A. and Raje, M.: MAPbox: Using Parameterized Behavior Classes to Confine Untrusted Applications, *Proc. 9th USENIX Security Symposium* (2000).
2) Alexandrov, A., Ibel, M., Schauser, K.E. and Scheima, C.J.: Extending the Operation System at the User Level: The Ufo Global File System, *USENIX 1997 Annual Technical Conference* (1997).
3) Bubenik, R. and Zwaenepoel, W.: Optimistic Make, *IEEE Trans. Comput.*, Vol.41, No.2, pp.207–217 (1992).
4) 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，情報処理学会研究報告，Vol.99, No.32, pp.83–88 (1999).
5) Mckusick, M.K., Bostic, K., Karels, M.J. and Quarterman, J.S.: *The Design and Implementation of the 4.4 BSD Operationg System*, Addison-Wesley Publishing Company, Inc. (1996).
6) 溝淵，新城，當眞，根路銘，喜屋武，翁長：投機的処理を支援するオペレーティング・システムにおける世界とプロセスの操作，情報処理学会研究会報告 95-OS-69-18, Vol.95, No.69, pp.103–108 (1995).
7) Santry, D.S., Feeley, M.J., Hunchinson, N.C., Veitch, A.C., Carton, R.W. and Ofir, J.: Deciding When to Forget in the Elephant File System, *17th ACM Symposium on Operating Systems Principles* (*SOSP99*), pp.110–123 (1999).
8) 品川，河野，高橋，益田：拡張コンポーネントのためのカーネルによる細粒度軽量保護ドメインの実現，情報処理学会論文誌，Vol.40, No.6, pp.2596–2606 (1999).
9) 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，pp.95–100 (1997).
10) Sun, J., Shinjo, Y. and Itano, K.: The Implementation of a Distributed File System Supporting the Parallel World Model, *Proc. 3rd International Workshop on Advanced Parallel Processing Technologies*, pp.43–47 (1999).
11) Sun Microsystems, Inc.: Sun OS Network Programming (1990).
12) 東村，松原，相河，加藤：モーバイルオブジェクトシステム Planet のプロテクション機構，日本ソフトウェア科学会第 1 回インターネットテクノロジーワークショップ（WIT '98）(1998).

付　　録

A.1 世界 OS サーバ内部の手続き

世界 OS サーバは，以下の 3 つのモジュールから構成されている．

- 世界モジュール
- ファイルモジュール
- プロセスモジュール

各モジュールは，**表 4** に示すような世界の操作（生成，削除，融合，内容一覧）のための手続きを提供する．世界モジュールにおける世界の操作のための手続きは，同名の世界 OS のシステムコールが発行されたときに呼び出される．これらの手続きは，表 4 に示したファイルモジュールとプロセスモジュールの対応する手続きを呼び出す．

表 4 に示した手続き以外の手続きを，以下に示す．

(1) 世界モジュールが提供する手続き

- `get_ancestors()`：祖先の世界のリストを返す．

(2) ファイルモジュールが提供する手続き

- `wfile_create()`：wfile オブジェクトを生成する．
- `wdir_create()`：wdir オブジェクトを生成する．
- `wfile_copy()`：子世界でファイルが更新されるとき，親世界から子世界にファイルをコピーする．
- `wdir_register()`：オブジェクト（wfile，または，wdir）を世界名前操作木に登録する．
- `wdir_unregister()`：オブジェクトを世界名前操作木から削除する．
- `wdir_lookup()`：指定されたファイル名に対応するオブジェクトを世界名前操作木から検索する．
- `wdir_getdents()`：世界名前操作木を探索して，`getdents()`（Solaris においてディレクトリエントリを取得するシステムコール）が必要とする結果を生成する．

(3) プロセスモジュールが提供する手続き

- `wproc_register()`：プロセスを，指定された世界の内容として登録する．
- `wproc_unregister()`：プロセスを，それが所属する世界の内容から削除する．

A.2 手続き `wdir_lookup` のアルゴリズム

手続き `wdir_lookup` の入力は，論理名，プロセスのカレントワーキングディレクトリ（以下 cwd），プロセスのルートディレクトリ，プロセスが所属する世界を示す WID および，利用者権限である．論理名は，cwd からの相対パスまたは絶対パスで表記される．cwd とルートディレクトリは，OID として指定される．これらの OID は，システムコールを発行したプロセスの wproc インスタンスから得られる．出力は，オブジェクト（wfile または wdir）の有無および，存在した場合はオブジェクトの OID と見つかった世界の WID である．

この探索は，最初に，世界モジュールの手続き `get_ancestors` を実行し，指定された世界の祖先を得る．次に，検索対象のオブジェクトの初期値を求める．これは，論理名の先頭文字が `"/"` ならプロセスのルートディレクトリ，それ以外は cwd とする．以上を得た後で，以下の手続きを繰り返し実行する．

(1) 論理名の中から，ベース名（論理名の中で `"/"` で区切られた文字列）を取り出す．ベース名が空であれば，検索対象のオブジェクトの OID と，それが見つかった世界の WID を返す．

(2) 取り出したベース名と，WID（祖先の世界のものを含む）の組を，検索対象のオブジェクト（この場合は必ず wdir オブジェクト）が持つディレクトリエントリのリストから探す．複数のディレクトリエントリが見つかった場合，入力と同じ WID を持つものを，それがなければ祖先の WID のリストの前にあるものを優先する．

表 4 世界 OS を実現する手続きのうち，世界の 4 つの操作に関連するもの
Table 4 The procedures that are related to the world operations in the World OS.

世界の操作	世界モジュール	ファイルモジュール	プロセスモジュール
世界の生成	`world_create()`	`wfile_world_create()`	`wproc_world_create()`
世界の削除	`world_delete()`	`wfile_world_delete()`	`wproc_world_delete()`
世界の融合	`world_merge()`	`wfile_world_merge()`	`wproc_world_merge()`
世界の内容一覧	`world_contents()`	`wfile_world_contents()`	`wproc_world_contents()`

(3) (2)でディレクトリエントリが見つかった場合，その操作を調べる．操作が ADD なら見つかったオブジェクトを次の検索対象のオブジェクトに指定して，(1)に戻る．操作が DEL ならファイルがないことを返す．

(4) (2)で見つからない場合，Unix のディレクトリ木を `stat()` システムコールで調べる．そこで見つかった場合，それに対応するオブジェクトを生成して世界名前操作木に登録する．そして登録したオブジェクトを次の検索対象のオブジェクトに指定して，(1)に戻る．見つからなかった場合，オブジェクトがないことを返す．

(平成 13 年 12 月 14 日受付)
(平成 14 年 4 月 16 日採録)

石井 孝衛（学生会員）

1977 年生．2000 年筑波大学第三学群情報学類卒業．2002 年同大学大学院修士課程理工学研究科理工学専攻修了．修士（工学）．

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．分散型オペレーティング・システム，オブジェクト指向，分散処理，並列処理に興味を持つ．1990 年情報処理学会第 40 回全国大会学術奨励賞，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．

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．