3T-1

異種ＯＳ相互運用方式の検討
－基本処理方式－

末広　亮太　　本田　邦夫
（松下電器産業（株）　東京研究所）

１．はじめに

半導体技術の急速な進展に伴い、マイクロプロセッサを用いたワークステーションあるいはパーソナルコンピュータの市場は急速な拡がりを見せている。最近では、ネットワークに接続された異機種・異ＯＳの環境においても高度なサービスを提供する分散システムの構築が注目されている。本稿では、ネットワークに接続された異機種の計算機が管理する資源を相互に運用するために、各種ファイルアクセスのためのシステムコールを、特定の機種、特定のＯＳに依存しない共通性の高い「システムコールピボット」という概念を基本にした方式を考案したので報告する[1]。

この方式は、異機種計算機間において、起動されたコマンドやアプリケーションプログラムなどが発行したシステムコールをピボットに変換し、このピボットをシステム間のメッセージの基本単位とすることにより、異種ＯＳ間の分散された資源を相互運用しようとするものである。

２．汎用ＯＳの分析

２．１　システムコールの分析と一般化

システムコールピボットの仕様を決定するため、各種ＯＳ[2],[3]のシステムコールの機能分析を以下に示す３つの面から行った。

① 機能面からの比較
② パラメータの比較
③ 各システムコールの実行に関わる状態遷移の比較

分析結果から、各種ＯＳ間で同じような機能を持つシステムコールでも、その機能や状態遷移等が正確に一致する例は少ないことを確認した。従って、各ＯＳのシステムコールをさらに詳細な機能単位に分割し、その機能単位を一般化する方法を採り、一般化した機能単位を「一般化機能単位」とし、この一般化機能単位を、

{ 制約条件 } 操作対象　の　操作基本機能

という一般的な表現で定義した。

２．２　ファイル格納形式の分析と一般化

システムコールピボットの操作対象となるファイルの概念を統一するため、ファイルの制御情報を以下の２つに分類し、各々について各種ＯＳがどのような情報を有するかを分析した。

① ファイル静的属性･･･ファイル生成日時、アクセス権など、ファイルを静的に管理する情報
② ファイル動的属性･･･ファイルポインタの位置、ファイル記述子など各プロセスが動的に管理する情報

３．ピボットの表現形式と処理単位

汎用ＯＳの分析と一般化の検討結果から、システムコールピボットに関する表現形式と処理単位の検討を行った。

３．１　ピボットとオブジェクト

ピボットは、各種資源に対応するオブジェクトのメソッドを起動するものと考え、以下のように定義した。

ピボット ≡ オブジェクトへのメッセージ

また、オブジェクトのクラスとしては、ファイルとディレクトリを想定し、これらの持つメソッドは、表２に示した一般化機能単位を割り当て、以下のように定義した。

メソッド ≡ 一般化機能単位 ＝ 操作基本機能＿操作対象＿条件

３．２　ピボットの処理単位

異機種間でのシステムコールﾚﾍﾞﾙの相互運用は、単一のメソッド、すなわち一般化機能単位を起動するだけでは実現できない。したがって、ピボットによって処理できる単位として、以下の３つを定義する。

① 単一処理 ･･ 単一のメソッドを実行
② 連続処理 ･･ 複数のメソッドをシーケンシャルに実行
③ フロー処理 ･･ 資源やシステムの状態により、実行すべきメソッドやメソッドの実行順序などが異なってくる処理

各種ＯＳのシステムコールは、一般化機能単位にもとづいた処理フローで表すことができるため、上記３つの処理単位は、すべてのシステムコールを表現するのに十分であると考えた。

３．３　ピボットの表現形式

ピボットは、オブジェクトに送信されるメッセージとして表現できるが、このメッセージは、上記の３つの処理単位に対応して、

① メッセージ式
② 連続メッセージ式
③ フローメッセージ式

の３種類の表現形式を取り得る。この表現形式として、ＢＮＦ記法に準じた各メッセージ式の抽象構文定義を行った。これを表１に示す。また、この抽象構文定義を用いて一般化機能単位を具体的なメッセージ表現に表したものを表２に示す。

A Study of Interoperability for Heterogeneous Operating Systems(Basic Architecture)
Ryota SUEHIRO, Kunio HONDA
MEI.Tokyo Research Laboratory

表1　ピボットの抽象構文定義（一部）

```
<ピボット>              ::= <メッセージ文>
<メッセージ文>          ::= <オブジェクト表現> <- <メッセージ>;
<オブジェクト表現>      ::= <オブジェクトクラス名> | <オブジェクト識別子>
<オブジェクトクラス名>  ::= File | StreamFile
<オブジェクト識別子>    ::= <オブジェクト変数> | <オブジェクト定数>
<オブジェクト変数>      ::= <変数>
<オブジェクト定数>      ::= <定数>
<メッセージ>            ::= <メッセージ式> | <連続メッセージ式> | <フローメッセージ式>
<メッセージ式>          ::= <単項メッセージ式> | <キーワードメッセージ式>
<単項メッセージ式>      ::= <機能単位セレクタ>
```

表2　一般化機能単位のメッセージ表現

No	一般化機能単位	ﾒｯｾｰｼﾞ表現
1	ファイルの存在の判定	File <- check_F_exist:path
2	ファイルの接続	File <- connect_F:path
3	ファイルの接続解除	ObjectId <- disconnect_F
4	ファイルのリンク	File <- link_F with:path
5	ファイルのアンリンク	File <- unlink_F
6	ファイルの移動	File <- move_F to:path
7	ファイルの静的属性の設定	File <- set_F_sattr of:sn put:mode
8	ファイルの静的属性の参照	File <- get_F_sattr of:sattrName
9	ファイルのアクセス権の判定	File <- check_F_accsRight by:mode
10	ストリームファイルの生成	File <- create_SF:path
11	ストリームファイルのオープン	File <- open_SF
≈	≈	≈

この表中の objectId は、抽象構文定義における＜オブジェクト識別子＞であり、これはクラスオブジェクト File または Directory から生成されたインスタンスオブジェクトを指す。

```
<< システムコール［ファイルの生成］>>
  pathで示されたファイルが既に存在する場合には、そのファイルをオープンし、存在しない場合は、modeでファイルを生成し、その後オープンする
-----------------------------------------------
file_create(path, mode)
{
    File <- check_F_exist:path;
    if(file == exist){
       File <- connect_F:path;
       File <- open_SF;
    }
    else{
       File <- create_SF:path;
       File <- set_F_sattr of:sn put:mode;
       File <- open_SF;
    }
}
```

図1　システムコールの抽象構文定義による表現例

以上のように、
① オブジェクトの持つメソッドの定義
② システムコールレベルの相互運用を実現するため、一般化機能単位の単一処理、連続処理、フロー処理の3種類の処理単位の採用
③ 前項の各処理単位について、それぞれピボットとしてのメッセージの表現形式の定義ならびに表現形式と処理単位を記述した基本仕様の作成を行った。

以上述べたような抽象構文定義を用いて、ある汎用ＯＳのシステムコールを表現したものが図1である。
この抽象構文定義により表現されたシステムコールを、コード化し、メッセージ形式に変換したものをシステムコールピボットと呼ぶ。

4．まとめ

以上述べたように、各種ＯＳのシステムコールをそのシステムコールのもつ機能単位を一般化することにより有効なピボットを定義できることを示した。

なお、この研究は、新エネルギー・産業技術総合開発機構（ＮＥＤＯ）が出資を受けて実施した、通商産業省工業技術院の大型プロジェクト「電子計算機相互運用データベースシステム」の研究開発の成果である。

5．参考文献

[1]末広 本田:"異種OS相互運用方式の検討-実験システムと評価-",情報処理学会第39回全国大会講演論文集
[2]"ＵＮＩＸ Ｓｙｓｔｅｍ Ｖ プログラマ・リファレンス・マニュアル",共立出版株式会社
[3]"ＭＳ－ＤＯＳ プログラマーズリファレンス・マニュアル",ＮＥＣ