6M-6

ＯＳインタフェース整合のための仮想化実現方式の一検討

松本 匡通　　金田 洋二
ＮＴＴ情報通信処理研究所

１．はじめに

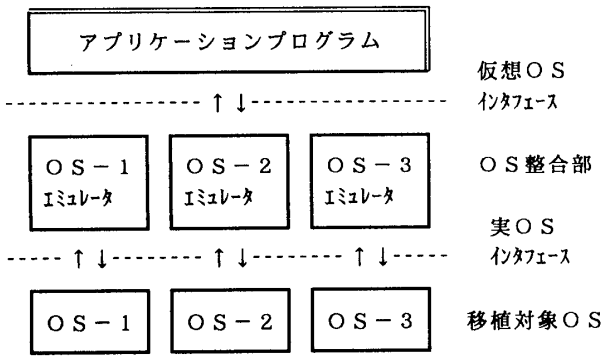
異機種コンピュータ間で、アプリケーション・プログラム（ＡＰ）やパッケージ・プログラム等のソフトウェアを移植する際、ＯＳインタフェースが異なり移植作業に工数を要する場合がある。このような場合に、ＯＳとＡＰの間に仮想化ＯＳインタフェースを設けＯＳインタフェースを整合することで、ソフトウェア移植を容易化する手法がある。本論文ではこの仮想化のための実現方式を提案する。

２．ＯＳインタフェース整合の必要性

ＡＰを異なるＯＳ環境に移植する場合、例えば、ＭＳ－ＤＯＳ上のＡＰをＵＮＩＸ上に乗せる場合等、ＡＰの走行環境の整合が必要となる。

このような移植の場合、通常はＡＰのＯＳシステムコールを記述している箇所を書直し、ＯＳインタフェースを整合させる作業を実施する。

このような作業を効率化するため、当初から多種類のＯＳへの搭載を考慮して、プログラム内のＯＳインタフェース記述部を１モジュールに集中させ、ＡＰ内の他のモジュールには仮想ＯＳインタフェースでＯＳ機能を見せる手法がある。この場合、仮想ＯＳインタフェースはＡＰ作成時の共通規約となる。現在、ＯＳインタフェースで標準化の努力が進められているものにＴＲＯＮ体系がある。その中で情報処理・通信処理分野向きで汎用的なＯＳ機能が整備されつつあるＣＴＲＯＮインタフェース*1を仮想ＯＳインタフェースにして、ＡＰのＯＳインタフェースを整備することができる。



図－１　ＯＳインタフェース整合の方法

ＯＳインタフェースの統一により、異種システムへの移植性の確保、プログラムの再利用や部品化といったソフトウェア開発技術上の課題の他、支援ツールや要員訓練といったプログラム開発体制の整備も容易となる。

３．ＯＳインタフェース整合の方法

(1)物理的なインタフェースの整合

ＯＳ毎に手続き呼び出しの方法（リンケージ・コンベンション）や、データ引継ぎ方法（メモリ配置条件、データ共有化条件）が異なる場合には、移植ＡＰを移植対象システムの言語処理系で再コンパイル・リンケージ処理することが必要となる。言語処理系では以下の変換処理を行なう。

①機械語コードの変更
②データ割付条件（データ写像規則）の変更
③割付アドレス情報の変更
④動作条件（メモリ保護条件、例外条件等）の変更

しかし、ＯＳコンベンション整合の為の実行時ルーチンの変更および共有データ域へのアクセスメソッドの変更等は、実行時に動的変換することが必要となる。

(2)構文規則の変換

システムコール名称やパラメータ名称の相違がある場合には構文規則の変換処理により整合することができる。移植対象のＡＰを、構文処理プロセッサ（例えば、ＹＡＣＣ，ＬＥＸ）で処理すれば、ソースレベルで変換できる。しかし、構文処理での静的変換では、実行時に決定するパラメータ内容の表現形式の変換等はできない。このような場合には、システムコールを動的に構文規則変換する方式を適用する必要がある。

(3)意味規則の変換

仮想ＯＳインタフェースと移植対象ＯＳインタフェースの相互間で、下記に示す意味規則的な不整合がある場合には、実行時に整合処理する必要がある。

①単一ＯＳ機能要素が複数のｼｽﾃﾑｺｰﾙに分割されたり、逆に複数のＯＳ機能要素が単一のｼｽﾃﾑｺｰﾙに集約されている。
②ｼｽﾃﾑｺｰﾙの呼出し順序が相違している。
③ＯＳ機能に過不足がある。

A Study on a Virtual Access Method for OS System-call Interface portability

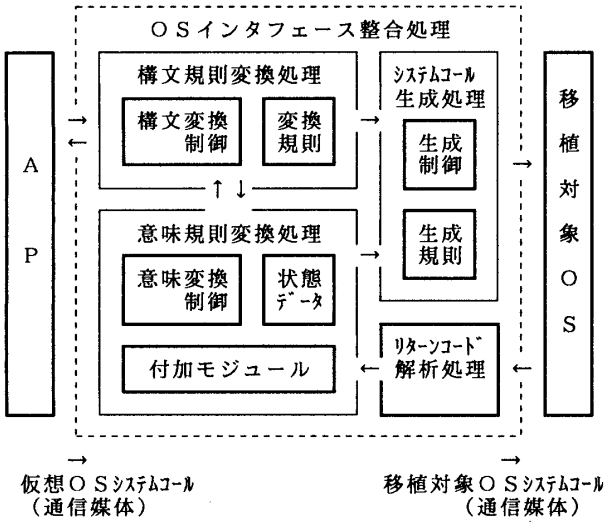
Masamichi Matsumoto, Yoji Kanada
NTT Communications and Information Processing Laboratories

4．ＯＳインタフェース整合化のための実現方式

従来、ソフトウェア移植のためのＯＳインタフェース整合は、ＡＰ内のＯＳシステムコールの変更やＯＳ側のカバールーチンの付加を手作りで実施する等ケースバイケースに対処している例が多い。本論文では３章の整合方法に従った、オブジェクト指向的なアプローチでインタフェース整合を図る適用範囲の広い実現方式を報告する。

(1)方式構成

ＯＳインタフェース整合を、システムコールやパラメータの表現形式の相違に伴う構文規則変換処理、ＯＳ機能の実現度の相違に伴う意味規則変換処理、移植対象ＯＳへのシステムコール生成処理およびリターンコード解析処理に分離する。



図－２　ＯＳインタフェース整合方式の実現例

(2)構文規則変換処理

一般的な構文規則変換処理の実現方式として、テーブル駆動方式と手続き方式がある。しかし、以下の理由でテーブル駆動方式の方が適用性がある。

①変換規則はほとんど一意的に決定され、変換規則の仕様を決定表あるいは状態遷移表で定義できるためテーブル化することで作成時や仕様変更時の対処が容易になる。

②実行時のＯＳシステムコールは、多くのシステムがＯＳシステムコール番号とパラメータリストの形式を採用しており、番号情報等で直接テーブル検索することが出来る。このため手続き方式に較べ、テーブル駆動方式でも実行効率はそれほど低下せず、メモリ効率はむしろ良くなる。

構文規則変換は以下の手順で行なう。

手順-1:入力システムコール番号およびパラメータ・リストで変換規則テーブルを検索し、変換処理種別を得る。

手順-2:変換処理種別固有処理を行い、中間情報を生成する。

①パラメータ値変換が必要であれば写像規則種別を決定（数値変換、文字列変換、ビット列変換、アドレス変換等）

②意味規則変換が必要であれば意味変換規則種別を決定

③システムコール生成が直接可能なら生成規則種別を決定

手順-3:中間情報により意味規則変換処理またはシステムコール生成処理を起動する。

(3)意味規則変換処理

ＯＳ機能実現度の相違を仮想ＯＳと移植対象ＯＳ双方の状態変数で管理し、システムコールの呼出し順序の相違や機能レベルの相違を変換すると共に機能不足部分については、付加モジュールにより機能追加する。

手順-1:仮想ＯＳｼｽﾃﾑｺｰﾙが移植対象ＯＳでは複数のｼｽﾃﾑｺｰﾙとなる場合、ｼｽﾃﾑｺｰﾙの展開と順次制御を行なう。

手順-2:仮想ＯＳでの複数ｼｽﾃﾑｺｰﾙが移植対象ＯＳでは１ｼｽﾃﾑｺｰﾙの場合、ＡＰのＯＳｼｽﾃﾑｺｰﾙ発行の遷移状況の監視および移植対象ＯＳｼｽﾃﾑｺｰﾙの発行契機制御を行なう。

手順-3:移植対象ＯＳの機能不足時に付加モジュールを起動し仮想ＯＳｲﾝﾀﾌｪｰｽを擬似する。

手順-4:ｼｽﾃﾑｺｰﾙ生成処理を起動する。

(4)システムコール生成処理

構文規則変換処理および意味規則変換処理から起動され、中間情報から移植対象ＯＳｼｽﾃﾑｺｰﾙを生成する。生成は、各ｼｽﾃﾑｺｰﾙ対応の生成ﾃｰﾌﾞﾙを参照することで行なう。

(5)リターンコード解析処理

移植対象ＯＳからのリターン情報を解析し引き継ぐ。

5．解決すべき課題

現実の処理系ではモデル化時の検討に加え、以下の課題を考慮する必要がある。

①仮想ＯＳ資源（ＣＰＵリソース、メモリリソース、ハードアクセス経路、割込み・例外処理）の一元管理をする必要がある。本方式では、意味規則変換処理用の状態データを使用して資源状態の管理を行なう。

②ＯＳインタフェース整合処理が実行時にインタプリティブな動作をするため、本来のＯＳと比較し処理効率（実行ステップ、メモリ使用量）の劣化が予想される。ＡＰの動作に悪影響を与えない範囲内かの評価が必要となる。

③従来の手作り的な手法に比較し、本方式を採用することでどの程度の作成工数の削減が達成できるか見極め本方式の有効性を評価する必要がある。

④仮想ＯＳｲﾝﾀﾌｪｰｽの充足性／統一性を、複数の移植対象ＯＳへ移植して確認するとともに、ＡＰ側からも評価する。

6．おわりに

本稿では、ＡＰとＯＳのインタフェース整合のための処理系設計に役立つ処理構造を示すとともに、評価方法を示した。現在、本稿で示したＯＳインタフェース整合方式に基づき、処理系を設計中である。今後、本方式の有効性を評価してゆく計画である。尚、本稿を執筆するにあたってご助言・ご協力頂きました各位に深謝いたします。

参考文献

*1　坂村他：原典ＣＴＲＯＮ大系、社団法人トロン協会