2V-2

異種OS上におけるUNIXインタフェースの実現法

遠城 秀和

NTT電気通信研究所

1. はじめに

新規にオペレーティングシステム(OS)を作成する際、ユーティリティとして既存のソフトウェアを移植し、ソフトウェア資産の活用を図ることは、OSの開発効率の向上に有効である。[1]

既存ソフトウェアの移植を容易にするには、OSの基本インタフェースとなるシステムコールと、そのソフトウェアが使用するファイル名などの走行環境が同一であることが必要である。そのために、システムコールを、他のOS上でシミュレートする方法について検討を行った。

本報告では、あるシステムコールを、他のオペレーティングシステム上でシミュレートする方法のモデルについて述べる。さらに、実時間制御を指向したオペレーティングシステム上で、TSS系オペレーティングシステムであるUNIXが提供している種々のコマンドの走行を可能とするために実現した、UNIXシステムコールのシミュレート方法をモデルを用いて示す。

2. システムコールシミュレートモデル

システムコールは、ユーザプロセスとカーネルとの間の基本インタフェースである。システムコールを中心にしてOSを整理すると、ユーザプロセス、システムコールライブラリに相当するシステムコール入口、カーネルの主要な機能であるシステムコール処理部の3つから構成されるモデルで表現できる(図1)。

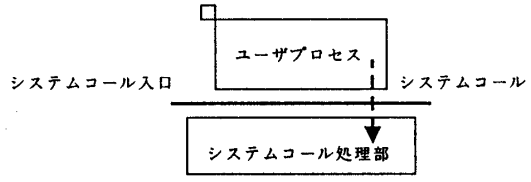


図1 オペレーティングシステムのモデル

このモデルにおけるシステムコールは、ユーザプロセスから依頼され、システムコール入口を通ってシステムコール処理部に制御が移された後、処理される機能と表現できる。

あるOS(実OS)上で別なOS(仮想OS)のシステムコール(仮想システムコール)をシミュレートするには、図1に示すモデルを基に表現すると、実OS上にシミュレートされるシステムコールのライブラリ(仮想システムコール入口)、仮想システムコールを使用するユーザプロセス(仮想ユーザプロセス)、仮想システムコールを処理するシステムコール処理部(仮想システムコール処理部)の3つを実現することになる。実現方法を整理すると以下のようになる。

① 仮想ユーザプロセスは実ユーザプロセス内で実現できる。

② 仮想システムコール処理部の実現には、次の3通りがある。
- (1) 仮想ユーザプロセスと同一の実ユーザプロセス内で実現する方法
- (2) 仮想ユーザプロセスと異なる実プロセス内で実現する方法
- (3) 実システムコール処理部内で実現する方法

③ 仮想システムコール入口の実現には、以下の2通りがある。
- (A) 仮想ユーザプロセスと同一の実ユーザプロセス内で実現する方法
  - 仮想システムコールは実ユーザプロセス内で変換され、仮想システムコール処理部との通信、すなわち(1)プロセス内通信(関数コール等)、(2)プロセス間通信(FIFOファイル等)、(3)プロセス・カーネル間通信(実システムコール)で伝達される
- (B) 実ユーザプロセス外で実現する方法
  - 仮想システムコールを実システムコールに追加してしまい、カーネルを経由してから仮想システムコール処理部に制御を移す

仮想システムコール処理部、仮想システムコール入口の実現方法を組み合わせると、システムコールシミュレートモデルとしては表1に示す6通りになる。

仮想システムコール処理部実現方法の定性的得失を表2に、仮想システムコール入口実現方法の定性的得失を表3に示す。

Simulation of UNIX Interface on another OS

Hidekazu ENJO
NTT Electrical Communications Laboratories

表1 システムコールのシミュレート方法

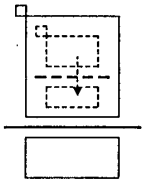
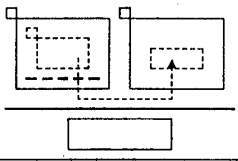
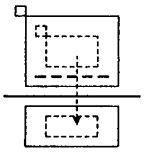
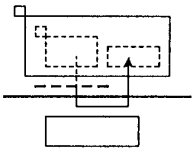
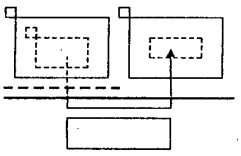
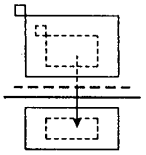
方法	モデル	仮想システムコール入口の実現位置	仮想システムコール処理部の実現位置
A-1		同一ユーザプロセス	同一ユーザプロセス
A-2		同一ユーザプロセス	他ユーザプロセス
A-3		同一ユーザプロセス	システムコール処理部
B-1		システムコール入口	同一ユーザプロセス
B-2		システムコール入口	他ユーザプロセス
B-3		システムコール入口	システムコール処理部

表2 仮想システムコール処理部実現方法の得失

方法(実現位置)	得失
(1) (同一ユーザプロセス)	・仮想システムコール処理部の全機能実現が必要 ・実システムコール処理部との独立性がある ・割り込み処理のタイミング等の完全性が保証不可 ・資源共有のためのカーネル内データのアクセスが困難
(2) (他ユーザプロセス)	・実システムコール処理部のプロセス、メモリ管理機能の流用可能 ・実システムコール処理部との独立性がある ・割り込み処理のタイミング等の完全性が保証不可 ・資源共有のためのカーネル内データのアクセスが困難
(3) (実システムコール処理部)	・実システムコール処理部の全機能の流用可能 ・実システムコール処理部への影響が大きい ・完全性の保証がしやすい ・資源共有がしやすい

表3 仮想システムコール入口実現方法の得失

方法(実現位置)	得失
A-1 (同一ユーザプロセス: プロセス内通信)	・リンク時のアドレス解決が必要
A-2 (同一ユーザプロセス: プロセス間通信)	・プロセス間での約束が複雑となる
A-3 (同一ユーザプロセス: プロセス・カーネル間通信)	・実システムコールが仮想システムコールのスーパーセットの必要がある
B (実システムコール入口)	・実OSとしても実現されている場合、APのバイナリレベルでの流用を可能としやすい

3. UNIXインタフェースの実現法

今回検討した実時間制御指向のオペレーティングシステム(実時間指向OS)のシステムコールとUNIXのシステムコールを比較すると次のようにまとめられる。

① 実時間指向OSのファイル操作のシステムコールは、機能及びインタフェースを含めてUNIXのシステムコールを包含している。

② 実時間指向OSのプロセス操作(プロセス間通信等)のシステムコールは、実時間性を実現するためUNIXとは異なる機能、インタフェースとなっている。

UNIXインタフェースの実現に対して、シミュレートの完全性と資源共有を要求する場合の実現方法は次のようになる。

UNIXのシステムコール入口の実現方法は、セマフォ、共用メモリ、メッセージ通信のプロセス間通信におけるアクセス権が、実時間指向OSのシステムコールでは包含できないため、実システムコールと仮想システムコールを共存させる方法Bとなる。

UNIXのカーネルの実現方法は、シミュレートするファイル操作系のシステムコールが実時間指向OSのシステムコールと共通している、カーネル機能をユーザプロセスで実現するとファイル等の資源共有が困難となるの理由から実時間指向OSのカーネル内で実現する方法B-3となる。

4. まとめ

既存のソフトウェア資産の活用を図りオペレーティングシステムの開発効率を向上させるため、他のオペレーティングシステムのシステムコールをシミュレートする方法を検討した。その結果、システムコールを他のオペレーティングシステムでシミュレートするシステムコールシミュレートモデルを提案し、UNIXシステムコールを実時間指向OS上でシミュレートする方法に適用した例を示した。

参考文献

[1]住永、牧、石井:
"日本語、リアルタイム、ネットワークなど機能拡張が進むUnix"
日経エレクトロニクス、1984年10月22号、pp.193-200