システムソフトウェアと
オペレーティング・システム 72－8
(1996. 5. 15)

OS／omicron V4 におけるハードウェアエミュレート方式とそれをサポートする言語処理系の設計と実現

○山本 康弘、早川 栄一、並木 美太郎、高橋 延匡

東京農工大学 工学部
東京都小金井市中町 2-24-16
0423-87-4600
yamamoto@omicron.ei.tuat.ac.jp
{hayakawa,namiki} @cc.tuat.ac.jp

あらまし　本稿では、OS／omicron V4 上に実現したハードウェアエミュレート方式と、それをサポートする言語処理系について述べる。計算機の使用目的の多様化に伴い、システムが多くの資源を扱うようになった。そのため、システムは目的に合わせ柔軟に構築できる必要がある。そこで筆者らは、ダイナミックリンクを利用することで、実行時にシステムを自由に構成し、ハードウェア固有のデータを実行環境に合わせて生成する環境を実現した。これにより、エミュレータ、言語処理系の構成を確立し、目的に応じて柔軟にシステム構成が行えるようになった。

キーワード　OS／omicron、エミュレータ、柔軟性、ダイナミックリンク

Design and Implementation of a Method of Hardware Emulation and a Language Processor for its Support

Yasuhiro YAMAMOTO, Eiichi HAYAKAWA, Mitarou NAMIKI, and Nobumasa TAKAHASHI

Faculty of Technology, Tokyo University of Agriculture and Technology
2-24-26 Naka－cho Koganei－shi－,Tokyo, 184 Japan
0423-87-4600
yamamoto@omicron.ei.tuat.ac.jp
{hayakawa,namiki} @cc.tuat.ac.jp

Abstract

This paper describes a method of hardware emulation on the OS/omicron, version 4, and a language processor for its support. With the growing diversity in the use of computers, Operating Systems (OS) must be able to handle many resources. Therefore, the system must be constructed with flexibility, and to match its intended use. We implemented a system which was constructed to provide flexibility at the time of execution through the use of a dynamic linking mechanism. Also, this system can generate the data peculiar to a hardware according to the execution environment. In this paper, the structure of emulator and language processor are established. This system can be constructed with flexibility to meet the user's use.

key words　OS／omicron、emulator、flexibility、dynamnic linking

1. はじめに

計算機の使用目的が多様化するにつれ、システムが扱わなければならない資源と同時に、データの形式も多様化の途をたどっている。

過去においては、浮動小数点演算ユニット(以下FPU)などに見られたように、システムをハードウェアを使用するものだけでなく、ソフトウェアエミュレータを使用して構築するような形があり、さらに、それぞれの表現形式においてデータ形式が異なってくるといった問題も存在した。[1] [2]

近年では、MPEG レコーダなどのマルチメディアデバイスにおいて、同様のケースが見られる。また、FPGA (Field Programable Gate Array) のようにユーザが容易にハードウェアを製作できるような環境も広まってきており、システムはより複雑な構成をとるようになることが考えられる。

このように、多様化する資源に対して、システムはハードウェア、ソフトウェアを問わず、柔軟に対応する必要がある。

筆者らは OS／omicron V4 [3] 上に、さまざまな資源を使用するシステムを柔軟に構成し、実行するための環境をエミュレータの形をとることで実現した。本稿では、OS／omicron V4 上のハードウェアエミュレータとそれをサポートする言語処理系について、実現した浮動小数点演算を例に設計と実現について述べる。

2. システムへの要求

(1) ルーチン呼出しのインタフェースの統一

FPU や MPEG レコーダを例にとると、これらはハードウェアで構築される実行環境、ソフトウェアエミュレータで構築される実行環境が存在する。これらの上で、同一モジュールの実行を行えるようにする場合には、まず、関数コールなど、ハードウェアとソフトウェアを呼び出すインタフェースを統一することが必要である。

(2) 柔軟にシステム構築できる OS

同一モジュールに対してさまざまなソフトウェアエミュレートを可能とする場合には、エミュレーションルーチンの中から必要なものを選択して、実行システムに組み込む必要がある。そこで、目的に合わせてハードウェア、エミュレーションルーチンなどを組み合わせて、システムの構築をできる OS が必要になる。

(3) ハードウェア固有データを変換できる環境

浮動小数点演算を例にとると、エミュレート時に必要となる定数をユーザプログラムにどのように持たせるかが問題となる。これを、実行コード中にビットパタンで持たせると、エミュレータを変更した場合に対応できなくなってしまう。そこで、定数を実行時に変換できる OS と、各形式に変換できる定数の表現形式を生成できる言語処理系が必要である。

3. システムの設計方針

2. で述べた要求から、システムの設計方針を次のように定めた。

(1) ユーザの使用目的に応じてシステムを動的に構築する

ユーザの使用目的によりプログラムのシステム構成は大きく、また頻繁に変更される。変更のたびに言語処理系やシステムの再構築を必要とした場合、モジュールの選択ミスなどによるリンクの失敗など予想外のバグを引き起こす恐れがあり、複数のバージョンが派生し、保守の手間も大きくなる。このため、システムはダイナミックリンクを利用し、実行時にソフトウェアシステムを動的に構築する。

(2) システム構成によりユーザプログラムのコードを変更しない

システム構成の変更により実行コードが変更されるようなシステムでは、プログラムの可搬性や、テストなどの開発効率は低下する。
このため、言語処理系が一定形式のコード

を生成し、エミュレータがシステム構成に応じてエミュレーションを行うことで、同一プログラムの、さまざまな環境上での実行を可能にする。

次章から、エミュレータを柔軟に構成するための OS の方式と言語処理系について述べる。

4. ハードウェアエミュレータの設計

4.1 エミュレート方式

ユーザプログラムとエミュレータの実装方式には次の 2 つの方式が考えられる。

(1) 静的リンク方式

プログラム中のハードウェアアクセス部分は、言語処理系の関数コール等の手続き呼出しの形を取り、エミュレータ部分をユーザプログラムと静的リンクすることでエミュレーションを行う。

この場合、言語処理系の生成する関数コールのオーバヘッドしか受けず、比較的高速にエミュレートプログラムを実行することが可能になる。しかし、OS やハードウェアに対する操作、一貫性を保つ機構が複雑になるといった問題がある。

(2) トラップ方式

ハードウェアアクセス部分は、OS に処理を移行するコードが生成され、OS 内部にあるエミュレータが起動されてエミュレーションを行う。

この場合、OS に処理が移行するので、コンテキストスイッチなど比較的大きなオーバヘッドが生じる可能性がある。

これら 2 つの比較を行った場合、(1) の方式では、ハードウェアを操作できる環境の構築や、システム構成の一貫性を保つことが困難である。(2) の方式は OS への処理移行に伴うオーバヘッドが問題になってくるが、OS／omicron V4 ではゲートコールを使用することで OS への処理移行に伴うオーバヘッドを関数コールと同程度にまで削減できる [4]。また、エミュレーションルーチンもダイナミックリンクを用いることで、動的に選択することができる。このことから、今回は (2) の方式で実現する。

4.2 システム構成

エミュレート方式は 4.1 で述べたように、トラップ方式を採用する。ユーザの使用目的に合わせ、動的にシステムを構築できるような構成を取るので、エミュレータ自体の構成はプロセッサの命令処理の過程を参考に次の (1) から (3) のように決定した。

また、同一コードでさまざまな環境に対応させるため、実行時にハードウェア固有のデータを生成する機構と、実行プログラムの一貫性を保つための管理部を用意した。全体構成を図 1 に示す。

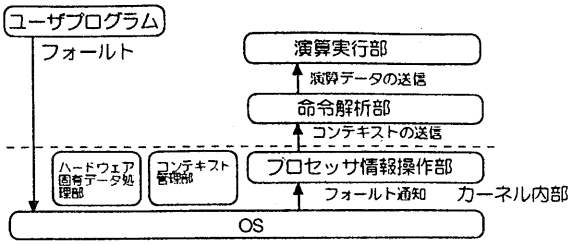

図 1 システムの全体構成

(1) プロセッサ情報管理部

プロセッサの命令フェッチ部分に当たる。ここでは、現在実行されているプログラムのプロセッサコンテキストの操作を行い、命令解析部、演算実行部を起動する。

ここは、システム構成の変化に影響されず、常に同一ルーチンが実行される。

(2) 命令解析部

プロセッサのデコード部分に当たる。ここでトラップを発生させたコードから解析を行ったり、ユーザプログラムから送られた値などを参照することで、演算の種類を判別し、演算の実行に必要な数値の取得、実効アドレスの算出などを行う。

実行コードの生成ルーチンを変更した場合などは、この部分のモジュールを変更することで対応する。

(3) 演算実行部

命令解析部から解析結果に従って各種演算を実行する。主に、この部分をユーザの使用目的に応じてダイナミックリンクすることで、さまざまな環境のエミュレートを行う。

(4) ハードウェア固有データ処理部

浮動小数点演算における定数などハードウェア固有のデータは、通常は実行コード中にビットパタンとして埋め込まれている。このような実行コードを、さまざまなハードウェアに対応させようとすると、必然的に実行コード中のデータを書き換えることになる。これは、保護などの点から考えると、危険な方法である。

このため、今回はハードウェア固有のデータはすべてテーブルを用いた間接参照を行う。テーブル中に格納されているデータは、最初ある一定の表現形式で格納されており、実行時にシステム構成が決定されてから、それぞれのハードウェア環境に対応したデータに変更する。(図2)

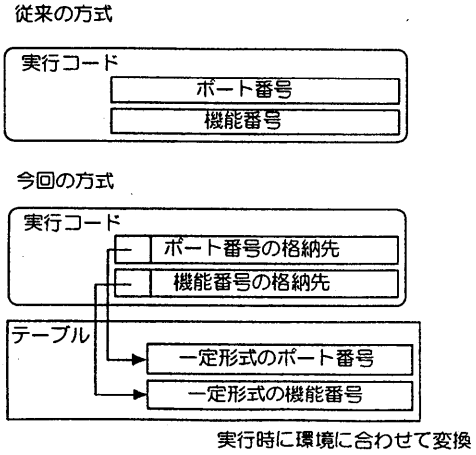

図2 ハードウェア固有データ処理

(5) コンテキスト管理部

ユーザプログラムはさまざまなシステム構成に対応しなければならない。そのため、実行されるプログラムは、環境の情報などを含んだ、拡張されたコンテキストを使用することで、その一環性を保つ。

このため、OS 内部のタスクやプロセス管理など、システム構成を変えられる最小単位の部分に、システム構成に応じて拡張したコンテキストを処理する部分を用意する。

5. 言語処理系の変更

5.1 言語処理系に対する要求

コード生成ルーチンでは 4. で述べたようなコードを生成できるように、言語処理系の変更を行う。このため、ハードウェア固有のデータを生成するためのテーブルの生成、トラップを発生させるコード生成を行う必要がある。

しかし、浮動小数点演算を例に挙げると、FPU を使用して高速化するために、IEEE の形態をすでに取った定数をコード中に埋め込むコードを生成する必要もある。また、エミュレート時のコード生成も、トラップ型の比較対象として、静的リンク方式を実現するためにコード生成を変更する場合などが考えられる。

このために、ある固定されたコード生成ルーチンのみを言語処理系でサポートするのではなく、ユーザの目的に合わせて言語処理系を構築する必要がある。

5.2 言語処理系の構成

言語処理系は、5.1 で述べたように、コード生成はユーザの目的に合わせて決定する。このため、コード生成部分から次のものをモジュール分けすることで、コード生成に柔軟性を持たせる。このときの言語処理系の構成を図3に示す。

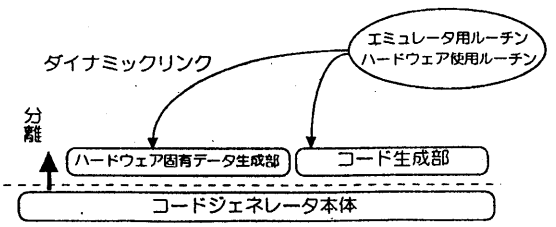

図3 言語処理系の構成

(1) 実行コード生成部

中間コードや、言語処理系中で作成した解析木などから、エミュレータの規則に従った実行コード、ハードウェアを直接使用するコードなどを生成する。

(2) ハードウェア固有データ生成部

ハードウェア固有のデータを作成する。ここで、エミュレート時のテーブルの生成、固有データの管理、テーブルへのアクセスのためのコードの生成を行う。また、ハードウェアを直接使用するようなコードを生成する時には、固定データをコード中へ埋め込む作業を行う。

選択に合ったモジュールをダイナミックリンクすることで、ユーザの使用目的に合った言語処理系を構築する。

6. 浮動小数点演算エミュレータの実現

ハードウェアエミュレータとして、今回は4. の設計に従って浮動小数点演算エミュレータを実現した。

6.1 実現環境

浮動小数点演算エミュレータを実現した環境を次に示す。

計算機　　: DELL OptiFlex 590
プロセッサ　: Intel Pentium 90MHz
OS　　: OS／omicron V4
言語処理系　: CAT386 [5] [6]

6.2 浮動小数点演算エミュレータの実現

表1に Intelx87 系 FPU に沿って作成した OS／omicron V4 上の浮動小数点演算エミュレータの実現規模を示す、また、表2に CAT386 の実現規模を示す。Intelx87 系はスタック型アーキテクチャを持ち80ビットで内部演算を行う FPU である。[7]

今回、エミュレート時のトラップは、Intelx86 系プロセッサ内の EM ビットをセットして FPU 用の命令は全て DNA 例外を発生させることで、OS にエミュレータの起動を通知する。これにより、エミュレート時とハードウェア使用時で同じコードで実行できる環境を構築した。

表1 浮動小数点演算エミュレータの実現規模

内容	新規記述行数
プロセッサ情報操作部	361行
命令解析部	1108行
演算実行部	3485行
計	4954行

表2 CAT386 の実現規模

内容	新規記述行数	無変更部
コードジェネレータ本体	220行	27000行
コード生成ルーチン	1989行	0行
定数生成ルーチン	582行	0行
計	2791行	27000行

6.3 ハードウェア固有データ処理

浮動小数点は複数の表現形式が存在しており、定数がそれぞれ表現形式固有のデータとして存在する。浮動小数点の定数データは、ダイナミックリンカを利用して最初のデータアクセス時に選択された表現形式に従ってビットパタンへの変換を行う。

テーブルに格納される浮動小数点の一定形式のデータは、ソース中に記述する形の文字列データを格納することで、各表現形式に対して変換できるようにした。

6.4 測定と評価

実現した浮動小数点演算エミュレータのいくつかの機能において、実行時間の測定を行った。この結果に、比較のため静的リンク型で実現したエミュレータの実行速度を付けたものを表3に示す。

表3 実行時間の測定

	フォールト型	関数コール型	ハードウェア
数値のロード（32Bit実数）	132μs	5μs	33.3ns
数値のストア（32Bit実数）	144μs	5μs	77.8ns
加算・減算	161μs	23μs	111.1ns
乗算	178μs	35μs	177.8ns
除算	312μs	174μs	811.1ns

ロードはアキュムレータスタックに、ストアはアキュムレータスタックから。四則演算はスタックトップとその下のものの計算。

フォールト型のものから静的リンク型のものを引いたものが、フォールト処理に要するオーバヘッドであり、これはほとんどがOS／omicron V4のコンテキストスイッチに要する時間である。

比較すると、単純な命令ほど大きな割合のオーバヘッドがある。しかし、今回の構成では、直接ハードウェアを使用するコードを使用して、ソフトウェアエミュレートを行うことができるため、このオーバヘッドは許容範囲内だと考える。

7. おわりに

本稿では、OS／omicron V4上に実現したハードウェアエミュレータと、それをサポートする言語処理系について述べた。本研究によって、OS／omicron V4上のエミュレータと言語処理系の構成を確立し、さまざまなハードウェア環境でのプログラムの実行、ハードウェアの検証などが行える環境を作成した。

今後は、ハードウェアが発生させる割込み情報をエミュレート対象のプログラムに通知する場合など、OSがエミュレータに対して操作を行うときの機構を設計、実現する。また、今回はFPUによる浮動小数点演算のエミュレートを実現したが、キーボードなど、それ以外のハードウェアエミュレータを実現する。

その後、仮想的なハードウェアインタフェースを持った仮想マシンなどを、この上に実装することで、エミュレータ内部のさらに細かいモジュール分け、作成したエミュレータの評価などを行っていく予定である。

参考文献

[1] 森岳志、他："OS／omicronにおける複数の浮動小数点方式のサポート"，情報処理学会第33回全国大会，pp.325－326，1986.

[2] 浜田穂積、他："万能実数値表現法 URRと OS"，情報処理学会コンピュータ・システムシンポジウム論文集，Vol.87,No.2 (1987)，pp.121－130

[3] Hayakawa, et.al:"Basic Design of SHOSHI Operating system that Supports Handwriting Interface"，情報処理学会論文誌，Vol.35，No.12

[4] 森永、他："2次元アドレスを管理するマイクロカーネルの実現と評価"，情報処理学会第50回全国大会論文集，3H－6，1995

[5] 中村浩之、他："80386用OS／omicron開発のための言語C処理系の実行環境の設計"，情報処理学会第44回全国大会，2F－10，1992

[6] 並木美太郎、他："2次元アドレスとダイナミックリンクのための実行コンテキストの設計と言語 C 処理系の開発"，システムソフトウェアとオペレーティングシステム 69-6，pp.31－36,1995

[7] i486™ マイクロプロセッサ プログラマーズ・リファレンス・マニュアル，インテルジャパン，1990

[8] McGrath, M.:"Virtual machine computing in an engneering environment"，IBM System Journal，Vol.11，No.2，pp. 199-218，1970