オペレーティング・システム　57－1
（１９９２．１２．４）

ＯＳデバッグ環境の設計と実現

清水正明、早川栄一、並木美太郎、高橋延匡
（東京農工大学　工学部　電子情報工学科）

本報告では、 OS のデバッグを支援するための環境である OS デバッグ環境について述べる。我々は、日本語情報処理を目的にして研究者が目的にあわせて自由に手を加えることのできる OS である OS／omicron の開発をおこなってきた。これらの OS の開発で、デバッグプログラムによってデバイスドライバの挙動が変化すること、デバッグ時に特権状態で動作する部分の保護や管理ができないことなどが問題になった。そこで、仮想計算機を使用してデバイスドライバのデバッグを容易にし、更にデバッグツールの構築を容易にする OS デバッグ環境を設計、実現した。

Design and Implementation of a Debugging Environment for Operating Systems

Masaaki SHIMIZU, Eiichi HAYAKAWA,
Mitarou NAMIKI and Nobumasa TAKAHASHI

Department of Computer Science,
Faculty of Technology,
Tokyo University of Agriculture and Technology,
Koganei-shi, Tokyo, 184 Japan

In this paper we describe an OS Debugging Environment which supports the debugging of OS's. With Japanese-language information processing in mind, we have developed OS/omicron, an OS whereby the researcher is able to freely enhance the system. In developing these OS's, we have found the following to be problematic: changes in the device-driver caused by the debugging program; and the inability to protect or control the sections operating under special privilage conditions. Employing a virtual machine, we have developed and then brought about the realization of an OS debugging enviroment which provides for the debugging of the device-driver, and makes simple the implementation of debugging tools.

1．はじめに

現在の計算機システムでは利用形態の多様化に伴って、多くの種類の資源を扱う必要が出てきている。例えば、ネットワーク上のサーバの場合、汎用の大規模のOSを使用するよりは、その目的に応じた小規模のOSを利用したくなる。実際にパーソナルコンピュータでは、さまざまなデバイスを接続して、個人用にシステムをコンフィグレートする利用形態が中心になっている。このようなシステムのパーソナル化に伴い、メニューによってシステム生成の設定を簡単に行うシステムも増えてきた。

しかし、ユーザインタフェース（以下、UI）やマルチメディアの研究のために、従来にないデバイスや異なる概念に基づく資源管理をシステムに導入しようとすると、機能が固定されたブラックボックスの従来のOSではなく、必要であればアプリケーションプログラマでさえも変更できるOSが望まれる。

このように計算機の利用形態の専用化に応じて、目的別にOSを生成する要求が生じている。

筆者の研究室では、日本語情報処理、UIの研究を行うシステムを実際に開発し、次のことが必要であるとの認識に至った。

・目的に合わせてOSを構成できること

OSが新しいデバイスに対応しやすい、つまり新しい資源管理プログラムを追加して再生成しやすい構成であることが必要である。我々はすでに、OS／omicron第2版［1］においてユーザ拡張部、OMICRON V3［2］においてデバイスドライバとソフトウェアバス、ハイパOS「礎」［3］においてソフトウェアバスで、目的別に機能が追加可能なOSの概念を実現し、実際に利用した。

・OSのテスト、デバッグが容易であること

目的別のOSを生成する場合、その資源管理などのデバッグが必要があるが、資源管理の場合、デバイスの割込みを扱うためにデバッグ、テストが困難である。これらの割込みを管理するプログラムを容易にデバッグできることが必要である。

・OSの設計の教育が必要であること

OSの内部構成、動作を理解することで開発、デバッグの効率を上げることができる。

そこで我々は、OSのデバッグやテストを容易に行えるOSのデバッグ環境を構築することにした。　本システムの特徴は次のとおりである。

（1）システム階層をAP層／OS層／デバッグ用仮想計算機層（ハイパOS層）に分けたことで、複数種類のOSのデバッグを可能にし、また、デバッグ環境の構築を容易にした。

（2）デバッグ用仮想計算機の機能として、メモリを仮想化、保護する機能であるVMMUを入れたことで、デバッガの保護、マルチプロセッサ環境のシミュレートを可能にした。

本稿では、このOSデバッグ環境の設計と実現について述べる。

2．OSにおけるデバッグの問題

アプリケーションプログラム（以下、OS）のデバッグを行う場合には、APの状態や使用している資源の情報をOSから得たり、操作したりすることができる。

逆に、管理する側であるOSのデバッグはAPのようにOSの機能を使うことはできない。さらに、OSは特権状態でハードウェア資源を管理するため、管理している情報の取得、保護が困難である。また、ハードウェアからの割込みによって非同期に動作するため、機構が複雑であることと、すべての条件でのテストが困難であることも問題になる。

我々は実際に、表示一体型液晶タブレット、プリントサーバのためのネットワークやレーザビームプリンタ、日本語処理のための仮名漢字変換などの処理機能を目的別のOSとして実現してきた。しかし、問題になったのは、そのOSのデバッグであった。

特に我々のシステムでは次のことが問題になった。

（1）目的別のOS生成においての問題

・メモリの不正アクセス

目的別に拡張した部分がOS核と同じ特権状態で動いているため、OS核やユーザプログラムを破壊した。

・プローブ効果

手書きインタフェースの研究のために表示一体型液晶タブレットを用いたシステムを構築した際に、デバッグにチェックプリントを使用したが、このプリント文を入れたことによるプローブ効果でプログラムの挙動が変化してしまった。この様に、デバッグ対象と同じ体系内でデ

バッグ用の機構を提供する手法は、デバッグ時にプローブ効果が顕著にあらわれるデバイスに対して問題がある。

・デバッグツールの流用ができない

OSにデバッグツールを静的にリンクした構成の場合、デバッグツールの拡張が困難である、OSから分離されていないので流用が難しいなどの問題がある。

・資源管理の状態が分からない

OSが管理している資源の確保や開放の状態を知ることができない。

(2) OS核においての問題

OS核で次のようなバグが発生した。

・メモリを二重に割り当ててしまう

・タスクの管理テーブル（TCB）の初期化ミスにより、タスク数限界付近のタスクを実行するとエラーが起こる

・メッセージ（send、receive）を発行するタイミングによってデッドロックが起こる

・資源のopen、closeのタイミングによってデッドロックが起こる

・ハードウェアの制限で、バイト単位でアクセスするとバスエラーになるI／O空間が存在した

また、我々は共有メモリ型マルチプロセッサマシンを対象とするOMICRON V3、ハイパOS「礎」の研究開発をすでに行い、さらに現在も並列／分散システムの研究を行っている。これらのシステムのシミュレートやOSのデバッグを行いたいという要求があるが、マルチプロセッサ環境でのデバッグにおいては、デバッガをどのプロセッサに置いて、どのように通信するかなどの難しい問題がある。

上記の問題から、OSのデバッグに次の問題意識を設定した。

(1) デバッグ環境に汎用性／拡張性が必要

(2) OSのメモリアクセスを監視（保護）できない

(3) OSの中からデバッグするとプローブ効果が発生する場合がある

(4) マルチプロセッサ環境でのメモリ管理や通信のデバッグが難しい

(5) OSが管理している資源の状態を知ることができない

(6) OSの内部動作の遷移を起こす割込みや特権命令の発生を知ることができない

3．OSデバッグ環境の設計

3．1 OSデバッグ環境の設計方針

第2章の問題意識から次の設計方針を定めた。

(1)「目的別OS」のデバッグ環境の構築が容易であること

特定のOS専用にOSデバッグ環境を作ると、OSデバッグ環境自体の開発やデバッグの手間が、そのOSデバッグ環境を使用することの利点を上回ってしまう場合がある。共通に利用できるデバッグ機能は共有すれば、多くのOSに対して容易にデバッグ可能にすることができる。

本OSデバッグ環境ではOSのデバッグ機能を、共通のデバッグ機能と、特定のOSに依存するデバッグ機能の二種類に分け、特定のOSに依存するデバッグ機能を取替え可能にすることにした。この構成にすることで多くのOSに対してデバッグ可能にすることが可能になる。また、一つのOSに対するデバッグツールも複数用意することができる。

(2) デバッグ機構がOSのバグにより破壊されないこと

OSのデバッグ機構がOSのバグによって破壊されないようにするには、OSとデバッグ用機能の資源管理を分離すべきである。本OSデバッグ環境では、OSのバグによりデバッグ機構への不当なメモリアクセスを防ぐため、メモリ管理機構を仮想化する。

(3) デバッグ機構がシステムの挙動に影響を与えないこと

プローブ効果が発生したのは、デバッグ対象とデバッグ用の機構を同じ体系内で提供したからである。プローブ効果を防ぐためには、OSとデバッグ用機能を分離すべきである。そのため、入出力デバイスと、その割込みを仮想化して割込みの取りこぼしをなくし、この仮想化の段階でデバッグ機能を提供することにした。

すると、OSはデバッグ用の機構を内部に用意する必要はなく、通常のサービスと同じ状態のOSをデバッグすることが可能になる。また、複雑な入出力操作を簡単に仮想化することで、OSでのプログラムを単純にし、バグの発生を少なくすることができる。

(4) 並列／分散システムのOSのデバッグを可能にすること

マルチプロセッサ／分散システムのOSのデバ

ッグを可能にするために、プロセッサ、入出力、メモリを仮想化し、複数に多重化して提供することで並列システムのシミュレートを可能にする。

(5) 特権状態で動いている部分もデバッグ可能できること

OS核は特権状態で実行されるため、デバッグや性能評価を行いにくい。デバッグ時には、特権状態や割込み禁止状態で動いている部分の動作を知りたい。OSデバッグ環境では(4)で述べたようにプロセッサ(特権状態)も仮想化することで、特権状態や割込み禁止状態で動いているOSをデバッグすることを可能にする。

(6) OSの状態遷移を監視／操作できること

OSは割込み、システムコールなどの状態遷移条件で内部動作が遷移するプログラムである。デバッグ時にはその状態遷移条件の発生を監視することができれば、そのタイミングをデバッグのきっかけとすることができる。また、発生した割込みのログを採り、あとで解析してデバッグやチューニングに役立てることが可能である。さらに、デバッグ時に任意のタイミングで割込みを発生できれば、きわどい条件でのOSの動作のテストを行うことが可能になる。割込みやプロセッサの仮想化の段階で、割込みの監視／操作機能、特権命令の監視などの機能を用意する。

3．2 本デバッグ環境の特徴

(1) システム階層をAP層／OS層／ハイパOS層に分けたこと

一つのデバッグプログラムにすべてのデバッグ機能を入れた場合、デバッグ機能の変更、デバッグするOSの変更などで、このデバッグプログラムの変更、再生成が必要になる。何度も変更／生成するのは効率が悪い上に、バグが混入するおそれもある。

そのため、入出力やプロセッサを仮想化する部分を仮想計算機というインタフェースでまとめ、ハイパOSで実現することにした。デバッグ機能に関しては、ハイパOSでは割込み、メモリアクセスの監視など共通の基本デバッグ機能のみを提供する。

また、仮想計算機上に具体的なOSデバッグ機能を持つデバッガを実行し、そのデバッガからハイパOSの基本デバッグ機能を使用するというデバッグ手法を採ることにする。このとき、デバッガとデバッグターゲットのOS(被デバッグOS)は別の仮想計算機上で実行されるようにすることで、デバッガの保護、OS独立性を確保する。

さらに、デバッガ側の仮想計算機上にOS(デバッガ実行用OS)を用意し、そのOSのAPとしてデバッグツール(OSデバッガ)を実行する。OSで入出力や共通のデバッグ機能を提供することでデバッグツールの構築を容易にすることができる。

このOSデバッグ環境の構成を図1に示す。

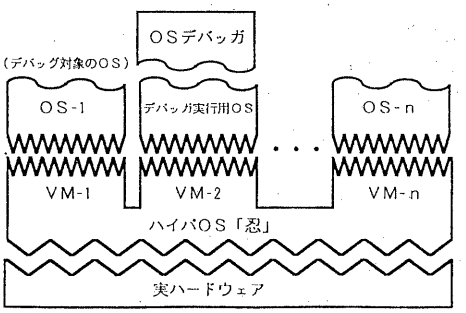

図1 OSデバッグ環境の構成

各層は次の機能を持つ

ハイパOS層……(ハイパOS「忍」)
- 複数の仮想計算機環境の提供
- 仮想計算機を操作する基本デバッグ機能の提供

OS層……(デバッガ実行用OS)
- OS独立のデバッグ機能の提供
- AP層に対する入出力の提供

AP層……(OSデバッガ)
- 特定OS／目的用のデバッグ機能の提供

この層に分けたことのメリットを次にまとめる。
- デバッグツールはOSの管理外に置かれるので、OSのバグで破壊されにくい
- デバッグ環境の一部が共有できる

また、層に分けたことのデメリットとしては、仮想計算機環境を提供することのオーバーヘッドがある。しかし、この問題に関しては、OSを安全かつ容易にデバッグすることの方が重要であるという立場をとる。

(2) 必要であれば入出力を仮想計算機で仮想化すること

OSとデバッグ用機能の資源管理を分離することで、OSの動作に極力影響を与えずにデバッグすることが可能になる。また、複雑な入出力操作を簡単に仮想化することで、OSでのプログラムを単純にし、バグの発生を少なくすることができる。

入出力を仮想化するメリットを次にまとめる。

・プローブ効果が少なくなる
・I/Oからの割込みを安全に拾うことができる
・多重化することでマルチプロセッサのデバッグを行える

例えば、我々は表示一体型液晶タブレットやディスク入出力を仮想化することで、デバッグを容易にし、またリアルタイム性を保証することができた。

仮想化のデメリットとしては、仮想化時のインタフェースに入出力形式が固定化されることがある。我々はSCSI、シリアルデバイスに関しては仮想化は行わずに操作することも許しているため、新しいデバイスの追加に利用可能である。

(3) VMMU機能を提供すること

OSデバッグ環境には、メモリ管理に関して次の要求がある。

・OSのバグでデバッグツールが破壊されないように保護したい
・デバッグ時に被デバッグOSのメモリアクセスを監視したい
・仮想記憶のOSのデバッグをしたい
・マルチプロセッサOSのメモリ配置のシミュレーションをしたい

これらの要求を満たすために、メモリやMMUを仮想化して提供することにした。

仮想化のオーバーヘッドによって速度の低下が問題になるが、この問題を低減するための機構については、第4章で提案する。

4．ハイパOS「忍」の設計と実現

4．1 ハイパOS「忍」の設計

第3章のOSデバッグ環境の設計から、「忍」を図2のように構成した。この設計の特徴について次に述べる。

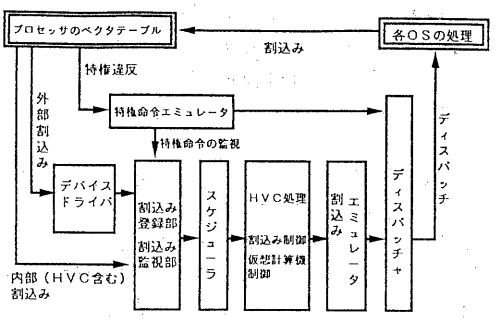

図2 ハイパOS「忍」の構成

4．1．1 プロセッサの仮想化

デバッグ時にプロセッサの状態を操作したり、命令を監視するためにプロセッサを仮想化し、管理する。

(1) 仮想計算機の状態を操作する

仮想計算機の状態には図3に示す5つの状態がある。デバッグ時に、デバッグ対象OSの仮想計算機をSuspended状態にすることで、デバッグ対象OSの動作を一時停止する。

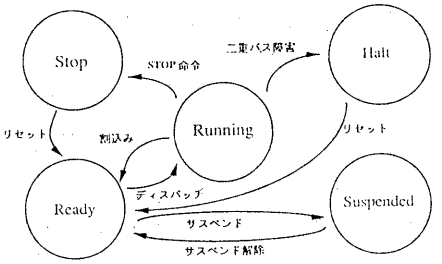

図3 仮想計算機のプロセッサ状態の遷移

(2) 仮想計算機のプロセッサのレジスタの操作

デバッグ時に、デバッグ対象OSの仮想計算機のプロセッサのレジスタを操作をすることができる。

(3) 特権命令の監視

OSはコンテキストスイッチや、排他区間操作時に特権命令を発行する。OSの動作切替えのタイミングを知るために、特権命令の発行を監視する。

4．1．2 割込みの仮想化、監視／発生

デバッグ時に割込みを監視、発生するために割込みを仮想化し、管理する。

(1) 内部、外部割込みの監視

OSの動作切替えのタイミングを知ったり、非同期の動作のログを取るために、割込みを監視できるようにした。監視するのは、外部（ハードウェア）割込みと、内部（ソフトウェア）割込みである。

(2) 内部、外部割込みの発生

OSの動作に再現性を出すために、指定の仮想計算機に対して割込みを発生できるようにした。デバッグ時に任意のシーケンスで、仮想的な外部割込みを発生することができる。

(1)、(2)の機能を実現するために、各仮想計算機の管理テーブルに図4のような割込みキュー

を用意し、割込みを管理している。デバッグ時に割込みを起こしたいときには、被デバッグOSの仮想計算機の割込みキューを操作する。また、監視していた割込みが発生したときには、被デバッグOSの割込みキューではなく、デバッグ用OSの割込み報告キューに割込みデータがつながれる。

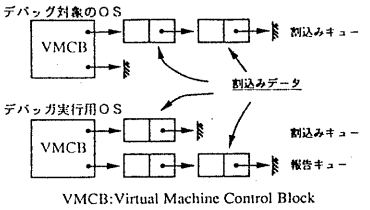

図4 割込みキュー、報告キューの構成

4．1．3 入出力デバイスの仮想化

ディスク、ビットマップを仮想化したウィンドウシステムなどの入出力デバイスに関しては、我々の研究室で開発された多重OS「江戸」[4]のデバイスドライバを利用している。表示一体型液晶タブレットのエコーバックや座標の仮想化、汎用のSCSIインタフェースの提供は今回行った。

4．1．4 メモリの仮想化

「忍」ではデバッグのために、物理メモリ、MMUを仮想化し、仮想MMUのインタフェースを提供する。また、MMUの仮想化のオーバーヘッドを減らすための機構を用意する。

(1) メモリ管理機構の仮想化

第3章2節のデバッグの要求を満たすために「忍」で物理メモリの管理を行い、仮想計算機に対しては仮想メモリ管理ユニット（VMMU）というインタフェースを提供する。

VMMUでは、ページディレクトリとページテーブルを使用して、1Gbyteの論理アドレスから64Mbyteの物理アドレスへの変換が可能である。また、ページ単位で読出し／書込み／実行のプロテクションを設定することができる。VMMUの初期設定で、複数の仮想計算機で物理空間を共有することができるため、共有メモリ型のマルチプロセッサマシンのシミュレートが可能である。

また、他のOSやハイパOSの破壊を避けるために、実ハードウェアのアドレス変換テーブルはOSには見せず、HyperVisorCall（以下、HVC）によってOSが用意したアドレス変換テーブルをハイパOS内のテーブルにコピーすることにした。このインタフェースにすることで、テーブルコピー時にアドレスチェックをすることと、OSに見せる実メモリを仮想化することが可能になった。物理メモリの使用例を図5に、アドレス変換の流れを図6に示す。

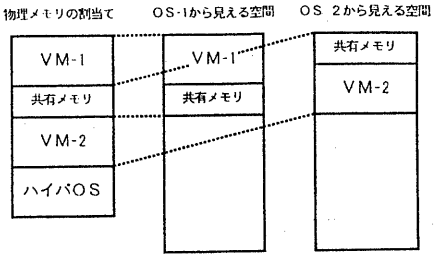

図5 共有メモリを持つメモリ使用例

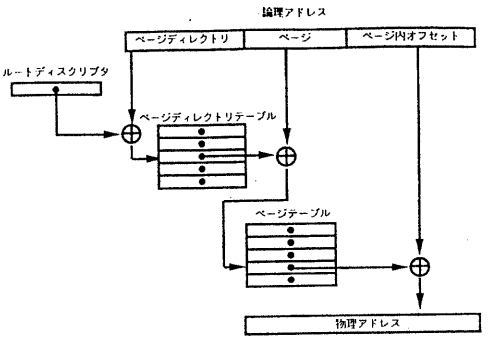

図6 アドレス変換の流れ

(2) 仮想化のオーバーヘッドを減らすための機構

MMUを仮想化し、アドレス変換テーブルをコピーする機構ではいくつかの仮想化のオーバーヘッドが発生する可能性があり、次のように対処している。

(a) 複数の仮想計算機切替え時のコピー時間

各仮想計算機ごとに変換テーブルのコピーを持つことで、仮想計算機切替え時にはテーブルの先頭を指すルートディスクリプタを切り替えるだけですむ。

(b) 仮想記憶OSで変換テーブルを頻繁に切り替えるときのコピー時間

変換テーブルを切り替えたときに、すべてをコピーするのではなく、ページテーブル単位でキャッシングする。つまり切り替えた直後はすべてのテーブルはInvalidになっており、そのアドレス

空間がアクセスされたときにフォールトしてコピーが行われる。この方法で、アクセスされない部分のテーブルコピーのオーバーヘッドをなくすことができる。

(c) 全アドレス空間分のページテーブルのコピーを持ったときのメモリ使用量

(b) とも関係するが、物理アドレス空間分のみの変換テーブルのコピー領域を用意して、キャッシングをして使うことでメモリ使用量を減らす。

(3) メモリアクセスの監視

OSデバッグ時にOSのメモリ管理のデバッグや、不当領域のアクセスを発見するために、デバッグ対象OSのメモリアクセスを監視できるようにした。この機能はVMMU機能と共に実現され、デバッグ対象のメモリに対して、一バイト単位で読出し／書込み／実行の監視（プロテクション）を設定することができる。プロテクションに違反すると、違反したOSにではなく監視を設定したデバッガにその違反が通知される。また、デバッグ時には、デバッグ対象の論理アドレス空間を、デバッガのメモリ空間ににマップすることができる（図7）。

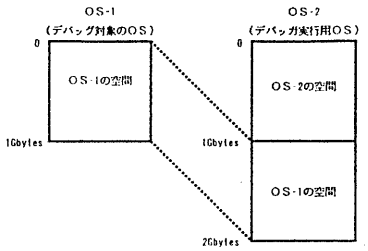

図7 デバッグ時のメモリマップ

4．2 ハイパOS「忍」の実現

ハイパOS「忍」には、OSデバッグのための基本機能がある。これらのデバッグ機能を、ハイパOSが提供する仮想計算機上のプログラムからのHVCとして使用可能である。HVCとしては状態を操作するもの、監視条件を設定するもの、の二種類を用意した。監視結果は割込みで通知するというインタフェースである。デバッグ用HVCの一覧を表1に示す。

「忍」は現在、日立製作所2050／32ワークステーション上で動作している。実現にあたっては、当研究室で開発され、OS／omicron第2版、OMICRON V3の実行環境として使用されている多重OS「江戸」のデバイスドライバを利用したことで、「江戸」とほぼ同じ環境を提供することができる。

表1 デバッグ用HVC一覧

HVC_set_debugger	デバッガ実行用ＯＳを登録する
HVC_get_OS_ID	ＯＳの名前からＩＤを得る
HVC_set_debuggee	デバッグ対象のＯＳを登録する
HVC_get_PRC_status	プロセッサ状態を得る
HVC_set_PRC_status	プロセッサ状態を設定する
HVC_get_CONTEXT	コンテキストを得る
HVC_set_CONTEXT	コンテキストを設定する
HVC_set_INTERRUPT	仮想割込みを登録する
HVC_set_INT_trap	割込みの監視を設定する
HVC_set_MEM_trap	メモリの監視を設定する
HVC_set_PSW_trap	特権命令の発行を監視する
HVC_get_INF_status	監視報告の情報を得る
HVC_set_TRACE	トレース状態にする

5．デバッガ実行用OSとOSデバッガ

仮想計算機上で実行されるデバッガを、デバッガ実行用OSとOSデバッガに分けたことは第3章で述べた。本章では、このデバッガ実行用OSとOSデバッガについて述べる。

5．1 デバッガ実行用OS

デバッガ実行用OSには、共通のデバッグ機能を提供することと、OSデバッガに入出力を提供するという二つの役割がある。共通のデバッグ機能としては、表2のSVCを考えている。このOSの実現方法としては、既存のOS／omicron第2版にこのSVCを付加する方法を考えている。

既存のOSを使用することで、開発環境を利用することができる。

表2 デバッガ実行用OSのデバッグ用SVC

仮想計算機状態の操作	
_SVC_set_VM_status	仮想計算機の状態を設定する
_SVC_get_VM_status	仮想計算機の状態を得る
_SVC_set_VM_context	コンテキストを設定する
_SVC_get_VM_context	コンテキストを得る
_SVC_set_VM_interrupt	仮想割込みを設定する
トラップ条件を設定する	
_SVC_wait_time	指定時間待つ
_SVC_trap_割込み発生	割込みの発生を監視する
_SVC_trap_割込み受付け	割込みの受付けを監視する
_SVC_trap_memory	メモリアクセスを監視する
_SVC_trace_proc	実行をトレースする
_SVC_set_breakpoint	ブレークポイントを設定する
条件が満たされるのを待つ	
_SVC_wait_trap	監視報告を待つ
結果を得る	
_SVC_get_trap_status	監視結果の情報を得る

5．2 OSデバッガ

デバッグの基本機能、入出力がデバッガ実行用OSから提供されるため、実現、プロトタイピングが容易にできることが特徴である。次のようなデバッガが考えられる。

(1) 対話型デバッガによる手動デバッグ
(2) バッチ型デバッガによるOSの自動チェック
(3) バッチ処理によるデモンストレーション

デバッガ実行用OSが用意されていない状態で、OSデバッガのサンプルとしてCPUモニタを作成した。OSがないため、すべてのデバイス、割込みの制御のプログラムが静的にリンクされている。

このCPUモニタでは、OS／omicron第2版の特権命令の発行数、コンテキストスイッチの回数、割込みの回数、さらにそれらのタイミングをビジュアルに示すことができた。

このツールでは、特にデバッグ機能は用意していないが、OSのコンテキストスイッチのタイミングや割込みマスクの状態がリアルタイムでビジュアルに表示されるため、OSの内部動作を知るのに役に立った。デバッガしては、第5章2節の(3)に入るであろう。このツールの実行画面を図8に示す。

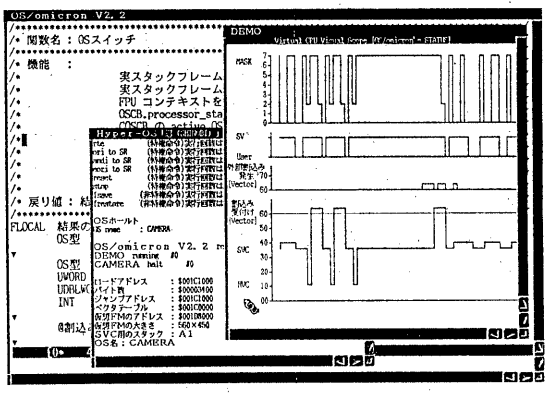

図8 CPUモニタ

6．OSデバッグ環境の実現

現在はハイパOS「忍」の実現がおわり、OSデバッグのテストをしている状態である。

性能評価の為に、ハイパOS「忍」の仮想化のオーバーヘッドの測定をした。測定結果を表3に示す。測定結果から、特権命令に20倍、割込みに200倍、ページテーブルのキャッシングなしの8Mbyteのメモリ空間の切替えで1．3msの仮想化によるオーバーヘッドがあることがわかった。

表3「忍」の仮想化のオーバーヘッド

仮想化した機能	ハイパOS「忍」	MC68020
特権命令(rte)	68μs	3．4μs
割込み	710μs	3．4μs
VMMU（8Mbyte切替え）	1．3ms	なし

また、ハイパOSの実現をモトローラMC68020プロセッサを使用したシステム上で行ったため、仮想計算機のプロセッサもMC68020仕様になっている。しかし、プロセッサや割込みの仮想化の概念は違うシステムにも利用可能である。

OSデバッグ環境の設計は終わっているものの、デバッガ実行用OS、OSデバッガを含めて、まだ完全に実現されてはいない。しかし、CPUモニタの試作で分かったことは、特権命令の発行タイミングや割込みの発生を監視できることは、OSの内部動作を知るために大いに役に立つと言うことがわかった。さらに、ビジュアルツールなどを使用するとOSの内部動作を直感的に理解させられることが分かった。また、ハイパOSやデバッガが完全に保護されるシステムであるため、OSを実際に作らせるなどの教育の場で使用できる。

7．おわりに

OSのデバッグやテストを容易に行える「OSデバッグ環境」について述べた。OSのデバッグだけではなく、この環境を利用して、OS設計の教育やデモンストレーションにも利用することを考えている。

参考文献

[1] 鈴木他：OS／omicron第2版の実現と評価、情報処理学会OS研究会、42－1、1989．
[2] 岡野他：並列処理用OSカーネル“Omicron V3”とハイパOSによる共有メモリ型マルチプロセッサへの実装、情報処理学会論文誌、Vol．32、No．6、1990．
[3] 並木他：ビルディング・ブロック・システムのための共有ソフトウェアバス～礎～、情報処理学会OS研究会、51－2、1991．
[4] 岡野他：多重OS「江戸」の設計と実現、情報処理学会論文誌、Vol．30、No．8、1989．