オペレーティング・システム　39－1
（１９８８．６．１７）

流通ソフトウェアを考慮したマルチOS

八木橋 信一*, 真鍋 和久**
* 日本電気(株) C&Cシステムインタフェース技術本部
** 日本電気技術情報システム開発(株)

本報告では、パーソナルコンピュータ用の流通ソフトウェアが利用できるUNIXシステムの実現方式について述べる。
多くのビジネス用流通ソフトウェアは、MS-DOS上で稼働するように作られているが、これらのほとんどは特定のマシンのBIOSルーチンやハードウェアに依存した機能を利用している。従って、マルチタスクのUNIX上で、MS-DOSやマシン依存の流通ソフトウェアが稼働できる環境を提供するためには、共有装置へのアクセスを制御する機能が必要になる。
このようなマルチOSの実現例として、日本電気のPC-9801シリーズパーソナルコンピュータ用のMS-DOSや流通ソフトウェアが稼働するUNIXシステムをPC-98XL²上に試作した。
このシステムは、マルチタスク環境で稼働するソフトウェアによる共有装置へのハードウェアアクセスを制御するために、装置のハードウェアエミュレーションを行う機能を持ち、マシン依存のソフトウェアの稼働を実用的な性能で実現している。

A Multiple Operating System for Existent Software

Shin-ichi YAGIHASHI *, Kazuhisa MANABE **
* C&C Systems Interface Engineering Laboratory, NEC Corporation
** NEC Scientific Information System Development Corporation
14-22, Shibaura 4-chome, Minato-ku, Tokyo 108, Japan

This paper summarizes implementation of a UNIX system that is capable of executing existent personal computer software, using a multiple operating system strategy.
Many business application software products run on MS-DOS, and most of them use BIOS routines or hardware dependent features of a specific machine. Thus an access control mechanism for sharing devices is an essential issue for implementing UNIX and MS-DOS multi-tasking environment.
A multiple operating system which simultaneously runs UNIX and MS-DOS application software has been developed on NEC PC-98XL² (X-L-double) personal computer, an 80386-based PC-9801 series machine.
The system has a hardware emulator for each sharing device so that software can access to them on the same level as a hardware under the control of the system. This system achieves reasonable performance even when the system executes machine dependent software.

1. はじめに

高性能マイクロプロセッサの普及により、パーソナルコンピュータのOS利用形態が変わりつつある。たとえばUNIXの利用形態は、ホストに端末を接続して利用する形態から、パーソナルUNIXマシンをネットワーク環境下で利用する形態へ移り変わりつつある。また、MS-DOSを利用しているソフトウェア開発者も、パーソナルコンピュータの高性能化とアプリケーションの巨大化にともない、OS/2やUNIXへの移行を検討するようになってきた。

ところで、このような移行の際に問題となるのは既存のソフトウェアの利用である。特にワードプロセッサやスプレッドシート(あるいはビジュアルなゲーム)に代表されるパーソナルコンピュータ上の膨大な量の流通ソフトウェアは、できればそのまま利用したい。この背景には、現在のUNIXで利用できるビジネスアプリケーションが日本語環境をも含めてかなり貧弱であるという要因もある。

この問題に対応するのが、Merge 386 (Locus Computing 社) や VP/ix (Interactive Systems 社と Phoenix Technologies 社)[1]で実現しているアプローチである。つまり、既存の流通ソフトウェアを稼働させることができるUNIXシステムの実現である。しかし、これらのシステムは米国のものであり、稼働機種はIBM PC-AT互換の80386マシンを対象としている。従って、これらをそのまま日本製のシステムに適用することはできない。

ここでは、日本国内で広く利用されている日本電気製のPC-9801シリーズパーソナルコンピュータを用いて、MS-DOSや流通ソフトウェアを稼働させることができるUNIXシステムの試作について報告する。[2],[3]

2. 稼働方式

まず、一般にマルチタスクOS上でMS-DOSや流通ソフトウェアを稼働させる方式について検討する。

複数のOSを稼働させる際には、共有する装置へのアクセスに気をつかう必要がある。同じ装置に対して複数のOSから同時にアクセスがあると、制御シーケンスが乱れ、正しくアクセスできなくなる。従って、次のような対応が必要である。

(1) OSを交互に切り換えて稼働させ、それぞれのOSが同時には装置を利用できないようにする。

(2) どちらかのOSによる装置へのアクセスを他方のOSが監視して制御する。被監視側のOSが装置へのアクセスを行うと、監視側のOSがこれを捉え、アクセスを代行したりエミュレートしたりする。

ここでは、(1)を「スタンバイ方式」、(2)を「エミュレーション方式」と呼ぶことにする。

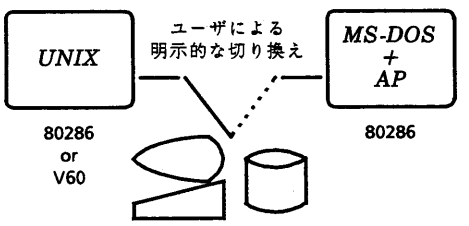

図2.1 スタンバイ方式

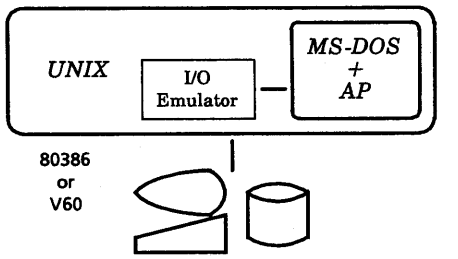

図2.2 エミュレーション方式

2.1 スタンバイ方式

この方式は、それぞれのOSの稼働空間を物理的かつ時間的に分離し、利用者からの明示的な切り換え要求に従ってOSを切り換える方式である。どちらかのOSが稼働している間は他のOSは停止しているため、それぞれのOSは他方を意識しない環境で稼働することができる。ただし、OSを停止できないような利用形態(ネットワークのサーバなど)になっているシステムには適用できない。

たとえば、PC-9801本体に搭載された80286のリアルモード(8086モード)でMS-DOSを稼働させ、プロテクトモード(80286モード)やアドオンボード上のV60でUNIXを稼働させる場合などにこの方式が適用できる。(図2.1)

2.2 エミュレーション方式

この方式は、UNIXの1プロセスとしてMS-DOSや流通ソフトウェアを稼働させ、タイムスライスによってプロセッサや周辺装置を共有する方式である。これは、UNIXのマルチタスク環境下でMS-DOSや流通ソフトウェアを利用できるうえ、ネットワークインタフェースを介してファイルの共有を図ることも可能である。ただしプロセッサとしては、8086やV30の命令実行機能と入出力命令のトラップ機能を持つことが必須である。

たとえば、80386のプロテクトモード(80386モード)でUNIXを稼働させ、バーチャル8086モードでMS-DOSを稼働させる場合や、V60のネイティブモードでUNIXを稼働させ、V30エミュレーションモードでMS-DOSを稼働させる場合などにこの方式が適用できる。[2](図2.2)

3. 実現のためのアプローチ

前章で述べた各方式は、いずれも特徴があり、比較検討を行うのも興味深いが、今回は特に80386を用いたエミュレーション方式の実現について述べる。

3.1 エミュレーションレベル

エミュレーション方式によって流通ソフトウェアを稼働させる場合、周辺装置をUNIXと共有する関係上、流通ソフトウェアが発する入出力命令をトラップして制御する機構が必要になる。このとき、トラップしてエミュレーションを行う入出力階層をどこに設定するかでエミュレーションシステムの構造がかなり違ってくる。考えられる階層は次の3つである。

(1) MS-DOSシステムコールレベル
(2) BIOSコールレベル
(3) ハードウェアアクセスレベル

図3.1に、これらの階層の位置づけを示す。以下、これらについてそれぞれ説明する。

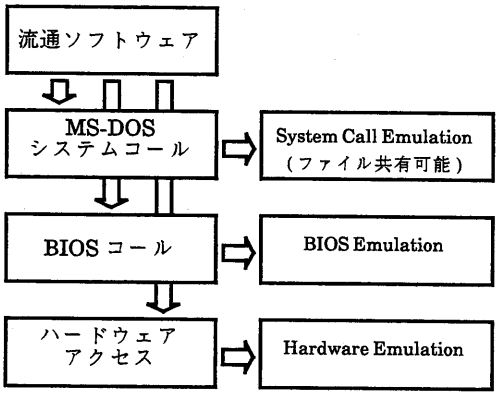

図3.1 エミュレーションレベル

3.1.1 システムコールレベルのトラップ

MS-DOSのシステムコールにフックして、システムコール自身をUNIXでエミュレートする方式である。この方式の問題点は、MS-DOSのシステムコールを完全にエミュレートすることの難しさと、システムコール以外の手段による入出力に対応できないことである。従って、この方式のみで稼働させることのできるソフトウェアはかなり限られる。しかし、ファイルアクセスを論理的なレベルで把握できるので、UNIXのファイルをMS-DOSのファイルとして扱うようなファイル共有機能が実現できる利点がある。

3.1.2 BIOSコールのトラップ

ROM BIOSにフックして、BIOSサービスをエミュレートする方式である。これは、比較的まとまった入出力操作を対象としてエミュレートするため、ハードウェアアクセスレベルでのトラップに比べて効率がよい。この実現には、UNIXのドライバに、ROM BIOSと等価なマルチタスク対応のサービスルーチンを用意する必要がある。

3.1.3 ハードウェアアクセスレベルのトラップ

エミュレーション対象となるソフトウェアが発するIN/OUT命令をトラップしてエミュレートする方式である。この方式では、ハードウェアレベルのアクセスから論理レベルのセマンティクスを解析して、装置の排他制御を行う必要がある。また、トラップの回数が比較的多くなるためオーバヘッドが高いという短所があるが、より多くのソフトウェアを稼働させることができる。

4. 試作システムの構築

ここでは、PC-9801用のソフトウェアを対象としたエミュレーションシステムについて述べる。具体的には、PC-98XL²(ノーマルモード)で稼働するPC-UX/V Release 3.0(80386用UNIX SystemV Release 3.0)上で、エミュレーションによってMS-DOSおよび流通ソフトウェアを稼働させることを目的としたシステムの実現方式について述べる。

4.1 エミュレーションレベルの選択

入出力をどのレベルでトラップするかは、エミュレーションの目的によって違ってくるが、できるだけ多くのソフトウェアに対応するためにはハードウェアアクセスレベルのトラップおよびエミュレーションが必要である。他の2つのレベルは、これと組み合わせて利用することができる。ハードウェアエミュレーションを、最適化したBIOSコールレベルのエミュレーションと組み合わせれば、BIOS経由の装置アクセスを高速化することができる。また、システムコールレベルのトラップメカニズムをファイルアクセスのリダイレクタとして利用すれば、UNIXファイルをMS-DOSの仮想ネットワークファイルとして利用することも可能になる。あるいは将来、Merge 386やVP/ixなどを移植する場合を仮定しても、ハードウェアエミュレーション機能は流用できると思われる。従って、まずハードウェアアクセスレベルのエミュレーションから実現することにした。

4.2 エミュレーションプロセス

今回試作したシステムについては、エミュレーション対象となるプロセス数を1つに制限した。MS-DOSはシングルタスクOSなので、1つのファイルシステムを同時に複数のMS-DOSが利用することはできない。従って、リダイレクタを介した仮想ファイルシステムを実現しない限りは、ファイルシステムを共有させることはできない。プロセス数を制限した理由は、ファイルシステムが共有できない場合、複数のエミュレーションプロセスを稼働させる意味が小さいことによる。

このエミュレーションプロセスは、80386のバーチャル8086モードで稼働するため、マジックナンバによって一般のUNIXプロセスと区別している。ファイルイメージは、ファイルヘッダとBIOS用のROMコードから構成される。このプロセスに対するメモリ割り付けなどの初期化処理、終了処理、およびトラップ処理は、一般のプロセスとは違った扱いになる。そのほかの処理、たとえばコンテキストスイッチングやスケジューリング、シグナルなどは、一般のプロセスと同じ扱いを受けるようにした。

4.3 メモリ空間

エミュレーションプロセスがアクセスできるメモリ空間は図4.1の通りである。

このうち、下位の640Kバイト(アドレス000000から09FFFFまでの空間)はスワッパブルなRAM空間であり、MS-DOSや流通ソフトウェアが利用する空間である。T-VRAM(テキスト用ビデオメモリ)空間は、エミュレーションプロセスがフォアグラウンドセッション(CRTとキーボードを占有した状態)かバックグラウンドセッションかによって割り付け方法が違っている。フォアグラウンドの時は実際のT-VRAMにトランスペアレントにマッピングするが、バックグラウンドの時は別のバッファ領域を割り当て、セッション切り換え(フォアグラウンドとバックグラウンドの交代操作)の際にT-VRAMの内容をコピーしている。G-VRAM(グラフィックス用ビデオメモリ)空間は、現在のところUNIXからのグラフィックスの利用を禁止しているため、常にトランスペアレントにマッピングされている。拡張ROM空間はトランスペアレントである。標準ROM空間は、コマンドファイルの内容をRAMにロード

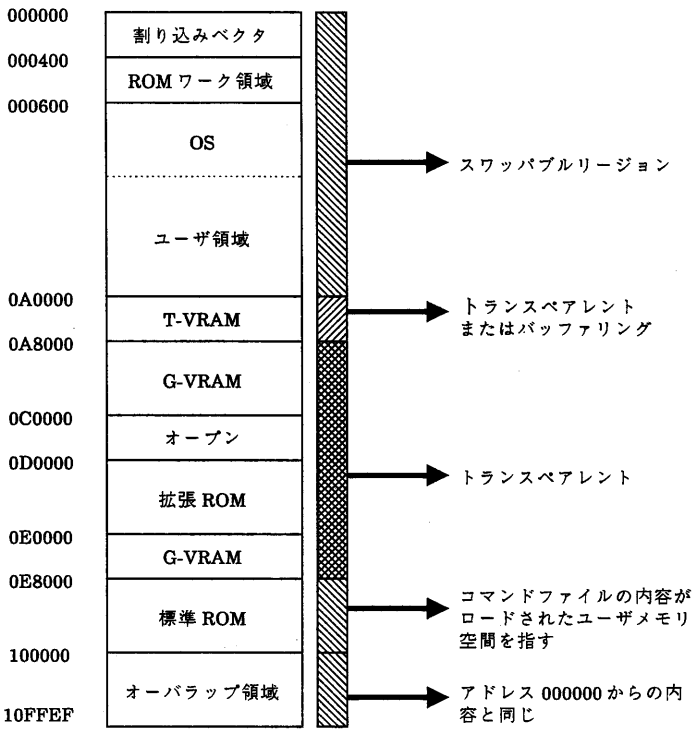

図 4-1 メモリレイアウト

して割り当てている。1M バイト以上のオーバラップ領域は、アドレスラップアラウンドをエミュレートするための空間であり、0 番地からの内容とオーバラップさせている。

4.4 例外処理

80386 では、バーチャル 8086 モードで稼働しているプロセスが、ある種の命令を実行しようとすると「一般保護例外」が起こる。このような命令には、INT や CLI などといったフラグを操作する特権命令と、IN や OUT のような入出力命令がある。(入出力命令については、選択的に透過させることもできる。)[4]

本システムでは、バーチャル 8086 モードの例外処理については特別な処理ルーチンを用意しており、トラップされた命令をデコードして命令の種類を判定している。これらの命令のうち特権命令については、プロテクトモードにかかわるものを除いてソフトウェアによって動作を代行し、プログラムの実行を継続できるようにしている。入出力命令については、入出力対象となる I/O アドレスからアクセス先の装置を判別し、それぞれ専用の装置エミュレータに制御を渡している。(図 4.2)

このほか、「コプロセッサ不在例外」に対しては、80387 数値演算コプロセッサが接続されているかどうかをプログラムが判定するために必要な命令に限りデコードし、NOP として扱っている。

4.5 装置のエミュレーション

UNIX と MS-DOS で共有する各装置については、アクセス中の排他制御を行う必要がある。従って、エミュレーションプロセスからの入出力命令は、すべてトラップされる。トラップ後の処理については、装置ごとにかなり違っている。たとえば、割り込みコントローラへのアクセスは、完全にソフトウェアエミュレートされ、実際の装置にアクセスすることはないが、ハードディスクへのアクセスは、最終的には装置に受け渡される。

多くの場合、プロセッサと装置のインタフェースや装置間のインタフェースは、ソフトウェアに

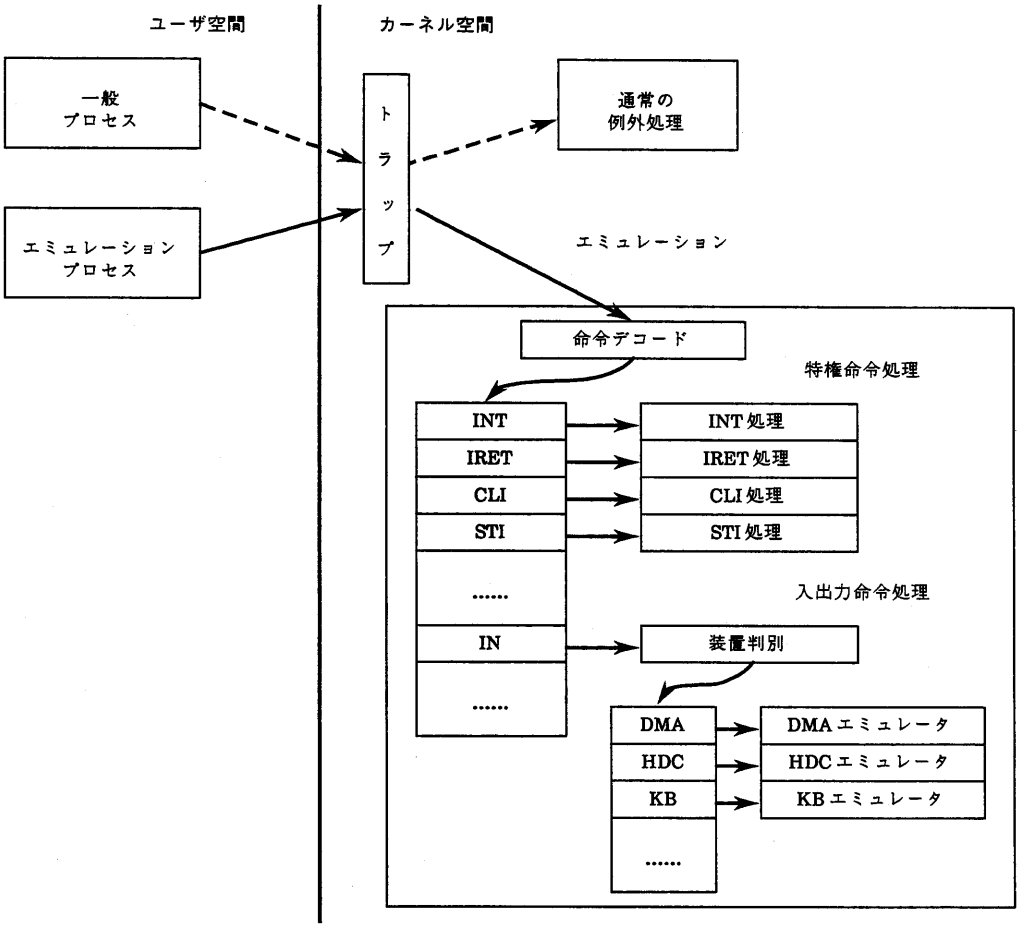

図 4-2 トラップ処理

よってエミュレートされる。このエミュレーション環境と実環境との違いは、稼働するプログラムによっては問題となることがある。たとえば、数ミリ秒単位での応答を必要とするリアルタイム性の高いプログラムや実行時間に依存するプログラム、またハードウェアのくせを巧妙に利用しているプログラムなどは、エミュレーション環境では正しく動作しないことが多い。

ここでは例として、ハードディスク周辺のエミュレーション環境を説明する。

ハードディスクコントローラに対するプログラムからの入出力指令は、アクセスに必要なすべてのコマンドとパラメタが揃うまでバッファリングされる。これは、エミュレーションプロセスが、ハードディスクを長時間占有したままになることを防ぐためである。このときハードディスクコントローラのエミュレータは、プログラムのステータス要求に対してハードディスクコントローラのエミュレータが仮定する状態に合ったステータスを返すようにしている。また、DMA コントローラのハードディスク用チャネルへのアクセスも、同様にバッファリングされる。すべてのパラメタが揃うと、ハードディスクコントローラのエミュレータは UNIX のハードディスクドライバに対してセマフォをとり、指示されたアクセスを行う。アクセス終了後には終了ステータスを取得し、セマフォを解放したのちに、割り込みコントローラエミュレータに対して疑似割り込み信号を送る。割り込みコントローラエミュレータは、マスクデータや他の割り込み要因を考慮してスケジューリングしたのち、プロセッサへの疑似割り込みを発する。その後システムは、エミュレーションプロセスに対して外部割り込みをエミュレートする。

これらの処理にかかわるものとして、DMA 転送の事前処理と事後処理がある。エミュレーションプ

ロセスのメモリ空間は80386のページング機能によって物理メモリ空間のあちこちにばらまかれているため、本システムでは物理的に連続した入出力用のバッファを用意している。このため、リード処理後とライト・ベリファイ処理前には、プロセスのメモリ空間とバッファとの間で転送処理を行っている。

4.6 ユーザインタフェース

本システムでは、画面上で4つまでのセッションを利用できる。セッションの選択は、キーボード操作(CTRL-f.1～CTRL-f.4またはSHIFT-CTRL-f.1～SHIFT-CTRL-f.4)によって行う。エミュレーションプロセスが稼働しているセッションが選択されると、エミュレーションプロセスはキーボードとCRTを排他的に利用する。システムは、エミュレーション環境の画面モードとUNIX環境の画面モードを把握しており、セッション切り換えの際に、それぞれの画面モードをCRTに設定している。

ある種のキーボード操作、たとえばSIGINTを発生させるためのキーやセッションを切り換えるためのキーボード操作などは、UNIXカーネルに渡す必要がある。本システムでは、以下のキーボード操作については例外的にエミュレーションプロセスに受け渡さず、UNIXカーネルが処理している。

- SHIFT-CTRL-STOP ..... エミュレーションプロセスにSIGINTを起こす。
- SHIFT-CTRL-COPY ..... 画面のハードコピールーチンを起動する。
- SHIFT-CTRL-f.1～SHIFT-CTRL-f.4 ..... フォアグラウンドセッションを選択する。

この例外処理を除けば、エミュレーション環境は実環境とほぼ同じユーザインタフェースを実現している。エミュレーションプロセスの起動は電源オンに対応し、プロセスの終了は電源オフに対応した利用イメージになる。

エミュレーションプロセスの起動はコマンド入力によって行い、終了はSHIFT-CTRL-STOPの入力によって行う。このほか、プロセスを終了させるシグナルを受けたとき、割り込み許可フラグ(IF)がオフのときのHLT命令の実行、およびプロテクトモード命令の実行によってもプロセスは終了する。

4.7 プロセス間インタフェース

本システムには、エミュレーション環境とUNIX環境との間に仮想的な全二重シリアルインタフェースが用意されている。

このインタフェースは、UNIXからもエミュレーション環境のMS-DOSからもキャラクタデバイスドライバによってアタッチされる。改行コードや日本語コードの変換を行うパイプを介して利用すれば、低速であるがOS間のテキスト転送が実現する。

これを利用することによって、たとえば、UNIXのプロファイラの出力結果をMS-DOS上で稼働するロータス1-2-3に転送し、編集してグラフ出力するといったことも可能になる。(図4.3)

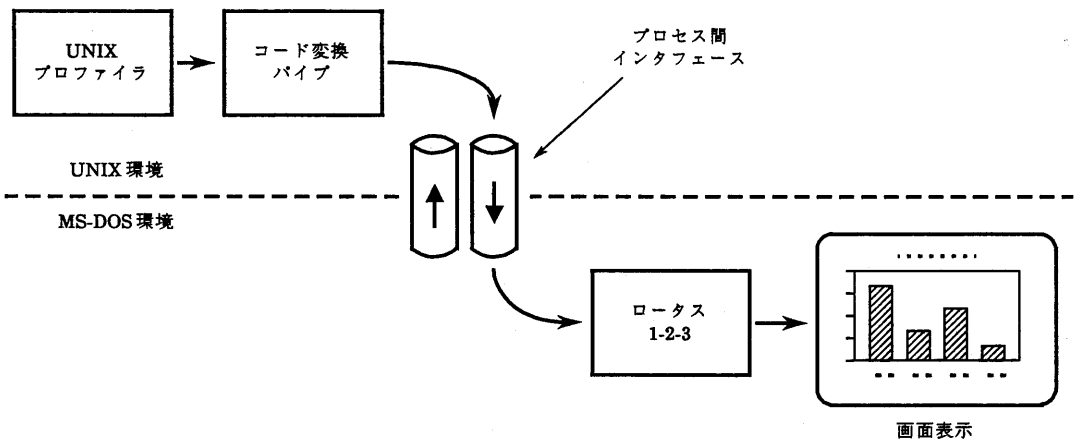

図4.3 プロセス間インタフェースの使用例

5. 評価

以上の結果、当初の目的であるUNIX上で流通ソフトウェアを稼働させるシステムを実現した。

本システム上で、動作を確認したソフトウェアとしては、「一太郎」や「新松」などの日本語ワードプロセッサ、「花子」や「鶴」などのグラフィックスエディタ、および「ロータス1-2-3」や「マルチプラン」といったスプレッドシートなどが挙げられる。

性能データとして、図5.1にベンチマーク結果を示しておく。全体的に、80386(リアルモード)の速

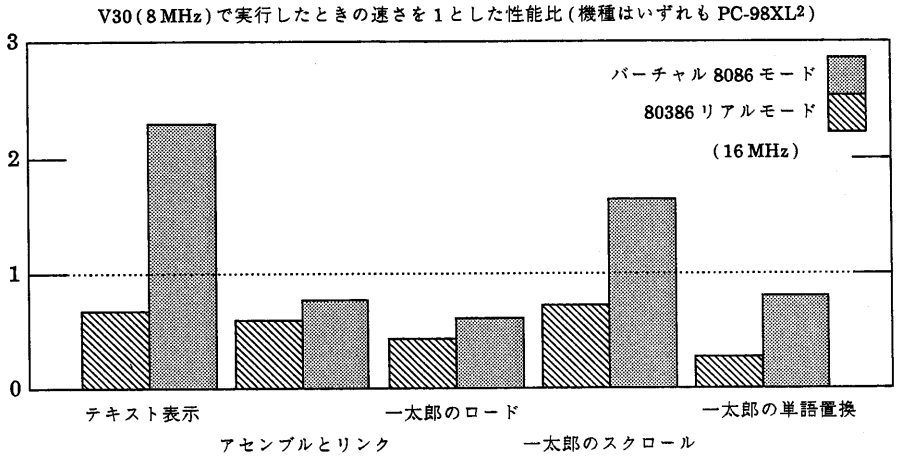

図5.1 ベンチマークテスト結果

さには及ばないが、通常の利用範囲内では、ほぼ実用的な性能がでていることを確認した。エミュレーションの要因別にみると、ディスク入出力の性能低下はあまり認められず、CRT表示関係のオーバヘッドが高いことがわかった。細かくみると、CRT表示でもビデオメモリに直接アクセスするソフトウェアのオーバヘッドは低いが、フォントをアクセスしたりGDC(グラフィックスディスプレイコントローラ)を利用するソフトウェアは、IN/OUTのトラップ回数が多い分だけオーバヘッドが高くなっている。

6. おわりに

流通ソフトウェアが利用できるUNIXシステムについて報告した。本システムの試作によって、UNIXプロセスとしてPC-9801シリーズ用の流通ソフトウェアが稼働することを確認した。また、通常の使用に耐えうる程度の性能は実現できた。

今後は、ファイル共有方式およびエミュレーションの改善について検討し、実現していきたい。

参考文献

[1] Judi Uttal : *Comparing VP/ix and Merge 386* ; UNIX WORLD vol.V, no.3, pp.75-82, 1988
[2] 八木橋, 真鍋 : 「V60マルチOSの実現」; 情報処理学会第36回全国大会, pp.357-358, 1988
[3] 三上 他 : 「複数のオペレーティングシステム環境とターゲットマシンリンクの開発」; 情報処理学会オペレーティング・システム研究会, 37-1, 1987
[4] Intel Corp. : *80386 Programmer's Reference Manual*, 1986
[5] アスキー出版局 : 「PC-9800シリーズテクニカルデータブック」, 1986