システムソフトウェアと
オペレーティング・システム 72−11
(1996. 5. 15)

オペレーティングシステム開発環境の設計と実装

安田 絹子† 久曽神 宏‡ 萩野 達也††

† 慶應義塾大学 政策・メディア研究科
‡ 慶應義塾大学 理工学研究科
†† 慶應義塾大学 環境情報学部
〒252 藤沢市遠藤 5322 0466-47-5111
{kinuko, kyu, hagino}@sfc.keio.ac.jp

あらまし オペレーティングシステムの安全で柔軟な開発環境として，既存のシステム上でオペレーティングシステムの実行を可能にするハードウェア・エミュレータ SESP (SPARC Emulator for Supervisor Programs) を提案する．SESP は SPARC のインストラクションセット・エミュレータ sparcemu を拡張・変更したものであり，オペレーティングシステムの動作に必要となる特権命令の実行，アドレス変換，デバイスなどをサポートしている．また，効率の良い開発のために，デバッグ機能や実行中の走行環境をファイルに退避する機能などを備えている．本論文では，SESP の設計と実装及び，実際に Mach 3.0 を使って行なった動作実験の結果について述べる．

キーワード オペレーティングシステム，開発環境，エミュレータ，デバッグ，メモリ管理ユニット

Design and Implementation of Development Environment of Operating Systems

Kinuko Yasuda† Hiroshi Kyusojin‡ Tatsuya Hagino††

† Graduate School of Media and Governance, Keio University
‡ Graduate School of Science and Technology, Keio University
†† Faculty of Environmental Information, Keio University
5322 Endou, Fujisawa city, 252, JAPAN +81-466-47-5111
{kinuko, kyu, hagino}@sfc.keio.ac.jp

Abstract For secure and flexible development environment for operating systems, we propose a hardware emulator SESP (SPARC Emulator for Supervisor Programs) that enables the execution of operating systems at the same level of user applications. SESP is an extension of sparcemu that is an instruction emulator for SPARC. SESP supports the emulation of privileged instructions, a memory management unit (MMU), and I/O devices. In addition, for efficient development, SESP has functions for debugging and dumping running environments on to files. This paper describes the design and implementation of SESP and the experiment of running Mach 3.0 on it.

key words operating system, development environment, emulator, debugging, MMU

1 はじめに

コンピュータ技術の急激な進歩に伴い，オペレーティングシステムの開発は次第に複雑なものになりつつある．しかし，オペレーティングシステムは一般的にハードウェア上で直接動作するものであるため，開発過程において安全な実験環境や充分なデバッグ機能を得ることは困難である．

Nachos [4] などの教育用オペレーティングシステムでは，学習者に安全な実験環境を提供する方法として，オペレーティングシステムとその動作環境を1つのユーザプロセスとして動作させるというアプローチをとっている．Nachos ではオペレーティングシステムの書き換え・実験にデバッガなどの様々な既存のツールを使うことが出来るが，オペレーティングシステム部分とマシン部分が一体となっているため，オペレーティングシステムの開発基盤としては位置付けられない．マシン部分のみをエミュレートするハードウェア・エミュレータも数多く研究されているが，その多くはユーザプログラムのプロファイリングやトレースを目的としており，オペレーティングシステムを動作させることは出来ない [5, 6].

本研究では，これらの問題点を踏まえた上で，ユーザプロセスとしてオペレーティングシステムを動作させることが可能なハードウェア・エミュレータ SESP (SPARC Emulator for Supervisor Programs) を提案する．また，実際に Mach を移植し，オペレーティングシステムの動作の確認と評価を行なった．

以下では，ハードウェア・エミュレータの設計及び実装，Mach の移植の実現方法，さらに性能評価について述べる．

2 背景

2.1 SPARC アーキテクチャ

SESP がエミュレートするハードウェア・アーキテクチャとして，SPARC アーキテクチャ[1] を採用した．SPARC アーキテクチャは Sun Microsystems で規定された RISC タイプのプロセッサであり，単純な命令形式，ウィンドウ化レジスタ，トラップテーブルによる高速なトラップハンドルなどの特徴を持つ．また，3 レベルのページテーブルを用いた物理アドレスから仮想アドレスへの変換など，汎用のメモリ管理機構を提供する MMU (Memory Management Unit) 参照アーキテクチャを定義している．

2.2 sparcemu

SESP のエミュレータ部分は sparcemu [2] をベースとしている．sparcemu はレジスタ群，メモリ，MMU，デバッグ機能などを持った SPARC のインストラクションセット・エミュレータで，条件コードの遅延評価，命令のデコード及びキャッシュなどによる高速化を行なっている．また，sparcemu は本来分散共有メモリ機構のハードウェアの実験ベースとして開発されたものであり，SPARC プロセッサのエミュレータ機能の他に Memory Based Processor (MBP) と COS (Comprehensive Operating System[8]) マイクロカーネルの機能を搭載している．

2.3 Mach

エミュレータ上で動作させるオペレーティングシステムとして，マイクロカーネルのソースコードがフリーで公開されてあり，(株) リコーによって SPARC に移植されたコードが利用可能であったことから，Mach 3.0 [3] を採用した．Mach は米国カーネギメロン大学で開発された分散オペレーティングシステムで，オペレーティングシステムとして必要な機能の多くをカーネル外部のユーザプログラムとし，マイクロカーネル上で動作するアプリケーションの単純化，機能拡張の簡易化を実現している．

3 設計と実装

SESP の設計は sparcemu の基本的な構成の上に必要な機能を追加・拡張するという形で行なった．

sparcemu は4つのプロセッサとそれらに共有されるメインメモリ，MBP，マイクロカーネルなどから構成され，それぞれのプロセッサはエミュレータ・モジュールのほかに MMU，オンチップ・キャッシュを持っている．このうち，SPARC アーキテクチャのエミュレートに必要のない MBP 及びマイクロカーネルのモジュールを切り分け，代わりにデバイスモジュールを加えた (図1参照)．

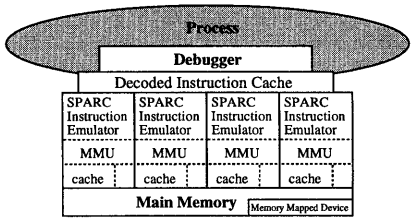

図 1: SESP の全体構成

また，オペレーティングシステムに対応するために，sparcemu でサポートされていなかった機能の追加，内蔵のマイクロカーネルによる使用を前提として設計されている部分の変更，デバッグ機能の拡張などを行なった．その他に，柔軟な開発環境の実現のために全体的にモジュール構成の見直しとモジュール性の強化を行ない，MMU，

TLB，デバイス，命令キャッシュなどの変更や削除，追加を容易に行なえるようにした．

3.1 命令の実行

SESP 及び sparcemu は，命令を実行する前に命令語を中間形式にデコードし，高速化のためにそれを Decoded Instruction Cache (dic) と呼ばれるキャッシュにストアする．また，命令の実行部分は明示的なループを用いず threaded execution という方法を取り，それぞれの命令のハンドラの最後で直接次の命令にジャンプしている．

sparcemu の dic は，ファイルのロード時にテキストセグメント中の全ての命令を保持できるだけの領域を確保し，デコードした命令をテキストセグメントと 1 対 1 で対応する形でストアしていく．このため sparcemu では次の命令へのポインタとして Program Counter Register (PC) の値ではなく dic 中のオフセットを用いている．しかし，この方法では最初にテキストセグメントへロードした命令以外実行することができない．また，次の命令へのポインタを dic 中のオフセットから得るため，UNIX 系のカーネルのように途中で PC を高位アドレスに移すコードを正常に実行することができない．

このため，SESP では dic を固定サイズとし，物理アドレスに対応するダイレクトマッピング方式でキャッシュするよう変更した．また，dic 中のオフセットを進める部分を全て PC に置き換え，PC に該当する dic 中のエントリを取ってくるルーチンを加えた．

3.2 MMU

sparcemu では MMU 及び TLB は単なるアドレス変換ルーチンに過ぎなかったが，SESP では内部レジスタと関連関数を集めて 1 つのクラスとして設計し直し，モジュール性の強化を行なった．また，sparcemu で省かれていたコンテキストテーブル，連続メモリ領域へのマッピング，内部レジスタへの書き込みによる MMU や TLB の制御などをサポートした．

3.3 デバッグ・性能測定機能

• **デバッグ機能**　sparcemu はアセンブラレベルでの実行のトレースやプログラム実行中のレジスタ・メモリの参照及び設定，番地指定によるブレークポイントの設定などのデバッグ機能を持っている．これを巨大なプログラムの開発基盤としても実用可能なものにするため，シンボルテーブルを用いたデバッグ機能やトラップによる実行制御などを追加した．また，コンソールの初期化を行なう前のオペレーティングシステムにデバッグコード出力の手段を提供するため，特殊なデバッグコンソールを実装した．

• **性能測定機能**　ブレークポイントまでの命令数をカウントする簡単な性能測定機能を実装し，ハードウェア上では特別な機器を用いなければ行なうことの出来ない命令数のカウントを容易に行なえるようにした．

• **スナップショット機能**　エミュレータを開発環境に使用することの大きな問題点として，速度の遅さによる開発サイクルの停滞が挙げられる．これを回避するために，実行中のメモリイメージとレジスタイメージを指定したファイルに退避する機能を実装した．退避されたファイルを使うことで，同じところから何度も繰り返し実行を再開することが出来るため，膨大な命令の中からデバッグの場所を絞り込むのに大きな効果をもたらす．また，実行中にメモリやレジスタの値を書き換えることが出来るため，少しずつ条件を変えて動作させることも可能であり，デバッグにかかる時間を大幅に短縮することが出来る．

3.4 デバイス

オペレーティングシステムを動作させるために，仮想デバイスとしてメモリマップドデバイスをいくつか用意した．デバイスの制御はマップされたアドレスを監視することで行なう．また，デバイスを実装するために必要なデバイス管理モジュールを実装し，新たなデバイスの追加・変更を容易に行なえるようにした．SESP にはインターバルクロック，割り込み制御レジスタ，コンソール，ディスクなどの基本的なデバイスをサポートする予定だが，まずプロトタイプとしてインターバルクロックと割り込み制御レジスタの 2 つのチップを実装した．

また，オペレーティングシステムがこれらのデバイスをマップできるように，物理アドレスとサイズ，デバイスの発行する割り込みのレベルなどをデバイスの名前と共に管理し，この管理情報の先頭番地を起動時に特定のレジスタにセットするようにした．

3.5 その他の変更・拡張

• **トラップ**　sparcemu では例外やシステムコールが起こった場合は内蔵のトラップハンドラが起動されていたが，トラップテーブルが初期化された場合はテーブルにしたがってオペレーティングシステムの規定したトラップハンドラに処理を移すよう変更した．

• **カーネルのロード**　SESP は起動時に実行ファイルから実行開始アドレスを読み，そのアドレスをもとにテキスト部のロードと最初の PC の設定を行なう．しかし，オペレーティングシステムの実行開始アドレスは通常カーネル領域内の仮想アドレスを指しているため，そのまま使うことはできない．そこで，もし実行開始アドレスが

カーネル領域の先頭アドレス(カーネルベース)よりも高位だった場合は実行ファイルがオペレーティングシステムだと判断し，一般的なブートプログラムと同様に実行開始アドレスからカーネルベースを引いた値を使うようにした．

なお，sparcemu ではテキスト部をロードする番地と最初の PC の設定を SPARC のユーザプログラムの実行開始アドレスである 0x2020 に固定している．

• **ASI**　SPARC アーキテクチャでは，複数の異なるアドレス空間のアクセスのために 256 ある ASI (Address Space Identifier) のうちいくつかに推奨する割り当てを規定している．sparcemu ではメインメモリ以外のアドレス空間はほとんどサポートされていなかったが，SESP では ASI 空間の割り当てを表 1のように拡張した．

表 1: SESP の ASI 割り当て

ASI	機能
0x02	Control Space Access (限定サポート)
0x03	MMU のフラッシュ/プローブ
0x04	MMU レジスタ
0x06	MMU TLB 診断
0x20～0x2f	MMU 物理アドレス透過
0x30	デバッグ用コンソール
0x40～0x41	テンポラリスペース

• **オペレーティングシステムへの値の引渡し**　デバイス情報の他に，ブート時に最低限必要とされる物理メモリのサイズ，ブートの引数をオペレーティングシステムに受け渡すため，これらの値を特定のレジスタに代入するようにした．

4 Mach の移植

エミュレータ上でのオペレーティングシステムの動作確認を行なうため，SESP への Mach の移植を行なった．

移植前の SPARC 用 Mach は，ブート時に Sun のブートプログラムである Open Boot PROM のサポートを前提としている．Open Boot PROM は，システムユニットの電源が入れられると Integer Unit (IU) の初期化，ページテーブルの作成やレジスタ及び主記憶のテストを行なう Power-ON Self-Test (POST) の起動，システムハードウェアの初期化などを行なう．しかし，SESP 自体はこの Open Boot PROM の機能を提供していないため，Open Boot PROM の機能のうちブートに必要なものを Mach 上に実現する必要がある．これらのアセンブラコードの追加と，SESP でサポートされていないデバイスへのアクセス部分を削除するために，1,000 行程度のソースコードの変更を行なった．

4.1 ページテーブル

Open Boot PROM は，オペレーティングシステムをロードするために，後にカーネルがカーネル領域内に作るページテーブルとは別に小さなページテーブルを作る．移植前の Mach では MMU が有効化され仮想アドレスモードが立ち上がった状態でブートされることが前提となっていたため，ブート直後に Mach 自らがこのページテーブルを作成する必要がある．そこで，レジスタから SESP の物理メモリのサイズを得て高位アドレスから順にページテーブルを作成し (図 2参照)，作成したコンテキストテーブルの物理アドレスを Context Table Pointer Register (MCTP) に代入し，MMU を有効化して仮想アドレスモードへ移るための SPARC アセンブラルーチンを実装した．

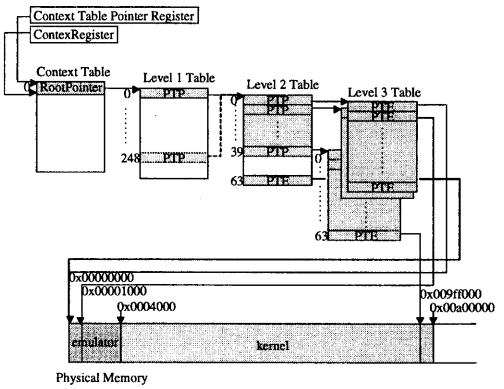

図 2: 実装したページテーブル

4.2 メモリの初期化

ブートの引数，デバイス情報など，SESP から受け渡すべき情報が全て Mach の管理するページテーブル上に移されると，メモリの初期化が行なわれる．

通常はカーネル領域外の全てのメモリがゼロクリアされるが，SESP はメモリ領域を必要に応じて確保しているので，ゼロクリアするためにメモリ領域を確保するという無駄な動作が起こってしまう．よって，SESP から Mach へ渡す情報が置かれていたページなど，Mach のページマップが管理していない，値受渡しに使っていた物理ページのみを不要領域としてゼロクリアするように変更した．

5 評価

5.1 オペレーティングシステムの動作状況

SESP 上でのオペレーティングシステムの動作実験を行なうために，前述したような書換えを行なった Mach を SESP 上に載せてみたところ，拡張あるいは変更した部分のうち，以下の各項目は完全に動作することがわかった．

- カーネルのメモリへのロード
- デバイス情報およびブート引数情報の引渡し
- 特権命令を含む命令の実行
- MMU によるアドレス変換
- オペレーティングシステムへのトラップ処理の転送
- カウンタタイマのハードウェア割り込み

ただし，現在 MMU のプローブ機能にバグがあることがわかっており，Mach の仮想メモリ管理部分の初期化の途中までしか動作できていない．実際には，Mach の 2,264,431 番目の命令である MMU のプローブ命令に失敗し，ブートの途中で止まってしまう．

なお，sparcemu 上で Mach を動作させた場合，サポートされていないアドレス空間へのアクセスのためにわずか 3 番目の命令でアボートする．これらのアドレス空間へのアクセスを一時的に全て取り除いて動かした場合，MMU の仮想アドレスモードを立ち上げるところまでは動作するが，その後仮想アドレス中の高位アドレスに飛ぶためのブランチ命令 (13,437 番目) で止まる．

5.2 dic の性能

SESP で新たに実装し直した dic の性能と PC を用いた threaded execution の動作を評価するため，命令実行部分のモジュールのみを入れ換え，10,000 回程度のループを含む単純なユーザプログラムを実行して比較を行なった (表 2参照)．

表 2: dic 及び命令実行の性能

	時間 (秒)	サイズ	キャッシュ方式
なし	6.93	—	—
SESP	2.67	固定	ダイレクトマッピング
sparcemu	2.32	可変	1 対 1

dic を使わない場合と使った場合では，約 3 倍程度の性能の差がある．また，sparcemu の dic と新たに実装したダイレクトマッピングの dic では，実装し直した dic の方がわずかに速い．

sparcemu では dic 中のオフセットを進めて命令を取ってくるため，PC の計算を毎回行なわない．しかし，オフセットの指しているエントリが初めて実行される命令だった場合，メモリ中から命令語を読み込んでデコードする必要があるため，オフセットから PC を計算し直さなければならない．この場合は，オフセットのインクリメントとオフセットから PC への変換との 2 つの処理が行なわれることになる．使用したプログラムの性質にも因ると思われるが，今回 SESP の方がわずかに高速だったのはこのためと考えられる．

一方，ユーザプログラムではなくオペレーティングシステムなどを動作させた場合，仮想アドレスモードを使用するため，今回実装したような物理アドレスによるダイレクトマッピング方式では 1 命令ごとに毎回 PC を 物理アドレスに変換する必要がある．従って，ユーザプロセスの場合よりも性能が落ちることが予想される．

現在，SESP の dic は平均 98~99%程度のヒット率で動作している．コンテキスト・スイッチが頻繁に起こるようになると PC の参照局所性が低くなるためにヒット率が低下すると思われるが，これは現在の実装状況ではまだ確認できていない．

5.3 sparcemu との性能比較

sparcemu と SESP に同じユーザプログラムを載せ，実行にかかる時間を計測して性能比較を行なったところ，SESP は sparcemu の約 2~4 倍程度の速さで動作することが分かった．高速化した理由は，ユーザプログラムではオペレーティングシステムのために拡張した機能がほとんど使われないこと，メモリ同期機構などのオーバーヘッドの大きい機能を取り外したことなどが考えられる．

表 3に，sparcemu との性能比較の結果を示す．また，各プログラムの主な命令の比率を表4に，各プログラムにおける dic のヒット率を表5に示す．実験に使用したプログラム及び環境は以下のようになっている．なお，各プログラムは全て `gcc -O3` でコンパイルした．

qsort	100 個の文字列のクイックソート
grep	約 500 行のファイルに対して GNU grep Ver. 2.0 を実行したもの
Native	SPARCstation 5 with 32MB memory, SunOS 4.1.4
SESP	デバイス及びシンボルテーブルサポートを取り外したもの
sparcemu	マイクロカーネルサポートを取り外したもの

表 3: sparcemu と SESP の性能比較

	qsort		grep	
	Time(msec)	Ratio	Time(msec)	Ratio
Native	6.8	—	1.5	—
SESP	125.1	18.4	63.9	42.6
sparcemu	295.3	43.4	235.8	157.2

表 4: 命令の内訳

qsort		grep	
命令	頻度(%)	命令	頻度(%)
ld	8.27	subcc	28.72
sethi	7.19	ldsb	11.25
subcc	6.83	add	8.01
srl	6.61	ld	4.11
subcc	5.80	bne	3.63
mov	5.72	ba	3.61
andcc	4.36	stb	3.57
bne	4.25	bcs	3.55
call	2.69	xor	3.36

表 5: dic のキャッシュヒット率

Program	Total	Miss	Hit (%)
qsort	341997	4418	98.71
grep	195124	209	99.89

5.4 既存のシステムとの比較

表6に，各システムでのユーザプログラムの動作パフォーマンスの比較を示す．表中の Performance は，ホストプロセッサに比べてかかる時間の割合を表している．

表 6: 各システムの Relative Performance

System	Target CPU	Performance	OS support
Shade[5]	SPARC	3~6	×
g88[7]	88000	30	○
Mable[6]	MIPS	20~80	×
sparcemu	SPARC	40~150	×
SESP	SPARC	18~50	○

Shade を除けば，SESP の実行速度は比較的高速といえる．最も高速な Shade は，Dynamic compilation と呼ばれる方法をとっている．この方法は非常に高速だが，しばしば1度に複数の命令をコンパイルするため，SESP の目標としているようなアーキテクチャの忠実なシミュレートには適当ではない．

ユーザプログラムによるテストでは拡張した機能のオーバーヘッドがほとんど含まれていないため，更にオペレーティングシステム実行時の性能測定と最適化を行なっていく予定である．

6 おわりに

本論文では，オペレーティングシステムに対応し，SPARC アーキテクチャの詳細なエミュレートを行なうハードウェア・エミュレータ SESP について述べた．sparcemu をベースにオペレーティングシステムの動作に必要な機能の拡張・変更を行ない，オペレーティングシステムの開発環境として実用性のあるエミュレータを実現した．

オペレーティングシステムの開発はハードウェア技術に大きく依存するため，常にハードウェア技術の開発から一歩遅れる形で行なわれている．SESP は既存の技術に捕らわれない自由な発想のもとでのオペレーティングシステムの開発支援を目的としている．また，各モジュールを柔軟に変更していくことが出来るため，ハードウェアとオペレーティングシステムの双方を互いの視点に立ってデザインしていくことを可能にする．

現在 SESP 上に Mach を載せて動作実験を行なっているが，特権命令の実行，メモリアドレス変換機構，トラップ処理などは完全に動作している．今後の課題として，デバイスの追加，高速化，仮想アドレスモードにおける各モジュールの充分な性能測定と最適化などが挙げられる．

参考文献

[1] SPARC International Inc. "The SPARC Architecture Manual, Version 8". Prentice Hall, Inc. 1992.

[2] SAITO Yasushi. A Fast SPARC Binary Emulator. Department of Information Science Technical Report.

[3] M. Acetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A New Kernel Foundation for UNIX Development", *Processing of the 1986 USENIX, Atlanta,* GA, 1986, 93-112.

[4] Wayne A. Christopher, Steven J. Procter, and Thomas E. Anderson. The Nachos Instructional Operating System. Computer Science Division, University of California Berkeley, 1994.

[5] Robert F. Cmelik and David keppel. Shade: A Fast Instruction-Set Simulator for Execution Profiling. Technical report, Department of Computer Science and Engineering, University of Washington, 1993.

[6] Peter Davies, Philippe Lacroute, John Heinlein and Mark Horowitz. Mable: A Technique for Efficient Machine Simulation. Technical Report, Stanford University, 1994.

[7] Robert Bedichek. Some Efficient Architecture Simulation Techniques. In *Proceedings of the Winter 1990 USENIX Conference,* Jan. 1990, p.53-63.

[8] N. Saito, H. Tokuda, T. Hagino, S. Oikawa, A. Yonezawa, S. Matsuoka, S. Inohara, Y. Tada, H. Sunahara, S. Ishii, E. Shibayama, and Y. Shinoda. "Comprehensive Operating System for Highly Parallel Machine", In *Proceedings of the 1994 International Symposium on Parallel Architecture, Algorithms and Networks (ISPAN),* pages 435-441, Kanazawa, Japan, December 1994., IEEE Computer Society.

盛光印刷所