システムソフトウエアと
オペレーティング・システム　60－23
マルチメディア通信と分散処理　61－23
（１９９３．　７．　９）

「紙」メタファを実現するための核ＯＳの基本設計

森永智之，早川栄一，並木美太郎，高橋延匡
（東京農工大学　工学研究科　電子情報工学専攻）

本報告では，紙メタファを提供するOS/omicron V4を構築するためのマイクロカーネルの基本設計について述べる．
手書き入力では紙のメタファを必要とするが，そのためには，紙のデータの多様性，高速なペンのレスポンスなどが重要となる．紙の持つデータの多様性のため，OSには様々な資源管理が要求される．
この資源管理の柔軟性を確保するために，我々はマイクロカーネル上で資源管理別のOSを動作させる方法をとる．さらに，これによってペンのレスポンスを保証することが可能となる．このマイクロカーネルを核OSと呼ぶ．核OSは，主としてプロセッサ管理，セグメントの管理，保護の機構を提供し，上位OSの実行環境を提供する．

A Basic design of a Kernel-OS for a paper metaphor

Tomoyuki Morinaga, Eiichi Hayakawa,
Mitarou Namiki and Nobumasa Takahashi

Department of Computer Science,
Tokyo University of Agriculture and Technology
2-24-16 Naka-cho, Koganei, Tokyo 184 JAPAN

This paper describes the basic design of a micro-kernel for the OS/omicronV4 which provides a paper metaphor.

Handwriting interfaces need paper metaphors, thus diversity of pen data and quick pen response become important. For diversity of pen data, various types of resource management are also needed.

We provide OS's for each resource in the micro-kernel to achieve flexiblity of resource management. In this approach, quick pen response is also guaranteed. We call the micro-kernel the Kernel-OS. The Kernel-OS provides an execution environment for OS's by managing the processor, segmentation and protection.

1．はじめに

我々の日常生活には，紙は必要不可欠である．紙には，アイディア，楽曲，地形といった様々な情報が，文字や記号などによって記録されている．例えば文章を作成する場合，文字だけでなく図や表を用いることが多い．図や表は，言語だけでは表せない直感的なイメージを伝えることができる．

もしも計算機で，文字，図，表といった様々な情報を制限なしに扱うことができるならば，文章作成の過程から，計算機の保存，再利用，検索といった特徴を利用できるようになる．これは，文章作成だけでなく，他の生産活動についてもいえることである．

我々は，文字以外の情報を計算機上で扱う手段として紙を仮想化することを提案し，研究を行っている．紙とペンを計算機に導入することによって，文字だけでなく，記号，絵などを統一的に扱うことができる[4]．

従来から，計算機上で文字以外の情報を扱う場合にはマウスを用いている．だが，マウスは単なるポインティングデバイスであり，紙とペンの感覚をユーザに提供することは難しい．しかし，表示一体型タブレットが登場したことによって，紙とペンをユーザインタフェースとして用いることができる可能性がでてきた．

このことから，現在我々の研究室では，手書き入力のための OS，OS/omicron V4（以下 V4 と表記する）を開発している[4, 5]．

紙を計算機上で仮想化する場合，OS 上の手書きソフトウェアに応じて，様々な紙の内部表現を定義し，それを OS に組み込み，OS のインタフェースとして用いることができれば有効である．このために，OS を拡張できる機能が必要である．この拡張の手段として，我々はダイナミックリンクを用いることにした．また，OS の機能拡張による保護の問題，紙の扱いが問題となってくる．これに対しては，2次元アドレスを用いることにした．手続きやデータをセグメントとして表現することによって，保護を行い，一枚の紙を論理的な単位として扱う．

このように，紙を計算機に導入することによって，OS では紙の属性管理，表示一体型液晶タブレットの管理といった様々な資源管理を行う必要が生じる．さらに V4 では，ユーザによって，様々な資源管理が付加される可能性がある．我々は，これらの資源管理をすべて単一の OS で行うことは難しいと考え，OS とハードウェアの間にマイクロカーネル層[6]を設け，この上で資源管理別に OS を実行することにした．このマイクロカーネルは OS を実行するための OS であり，以下ではこれを核 OS と呼ぶ．

本報告では，まず V4 について述べ，次に核 OS の設計について述べる．

2．OS/omicron V4

ここでは，OS/omicron V4 の概要について述べる．まず，紙を仮想化して計算機内に実現することを考えた場合，次の問題が生じる[4, 5]．

（1）紙の多様性

紙の利点は，その上になんでも書くことができるということである．この利点を活かすためには，仮想化された紙が書く対象を制限しないことが重要である．したがって，仮想化された紙の上には，文字，図形，表，記号などの様々なデータが表現されることになる．このような紙の多様性をユーザに提供することが重要である．また，個々のデータ構造を OS であらかじめ定義するのではなく，必要に応じて拡張できることが重要である．

（2）紙の多義性

前述したように，仮想化された紙は，書く対象を制限しないことが重要である．このために，仮

想化された紙の上には様々な情報が記録されることになり，紙に書かれたデータが文字としても，記号としても解釈可能な場合がある．このような，紙に書かれた元のデータは，文字認識，図形認識を経た後でも参照可能でなければならない．したがって，いったん紙に書かれたデータは，何らかの処理を経た後でも消えずに残っており，ユーザがこれを参照できるようにする．

（3）ペンのレスポンスの重要性

計算機で紙とペンを実現する場合，ペンのレスポンスは重要である．表示一体型タブレット上でペンを用いて文字を書いた場合，その筆点列のエコーバックに時間がかかると，思い通りの文字を書くことができなくなってしまう場合がある．このために，手書きインタフェースを用いる場合には，我々が実際の紙にペンで文字を書くようなリアルタイム性が重要である．

V4 では，以上の問題を解決するため，仮想化された紙として，「電紙」という概念を定義している．ここでは，紙を仮想化する V4 の設計と，電紙について簡単に述べる．詳しくは，文献[4, 5]を参照のこと．

（1）電紙の管理手法として二次元アドレスを用いる

電紙には，その内部情報が格納される．当然，システム内には複数の電紙が存在する．これらのことから，V4 では電紙を管理するために2次元アドレスを導入する．電紙をセグメントとして扱い，オフセットで電紙の内部データにアクセスする．これによって，仮想化された紙をシステム内でまとまった単位として扱うことが可能となる．また，保護の面からも有効であると考える．

（2）OS の機能拡張方式としてダイナミックリンクを用いる

電紙は，独自の内部表現を持つため，例えば電紙を可視化する場合には，電紙の属性によって可視化する手続きを変更することが必要になる．このように，属性によって手続きの実体を変える機構が必要となってくる．このために，V4 ではダイナミックリンク[1, 2]の機構を用いる．

（3）電紙のリンクを実現するためにワンレベルストアを用いる

V4 では，紙の属性変換という操作を通して，紙を加工していく．つまり，例えば手書き文字が書かれた紙に「文字認識」という属性変換を行うことで，コード化された文字が書かれた紙が生成されることになる．ここで，紙に書かれた文字を保存しておくという方針から，属性変換前と後の間で，リンクが生成されることになる．

電紙はリンクで強く関係付けされるので，これらのリンクを保存しておくことが必要となる．そこで，ファイルのインタフェースをメモリに統一することによって，2次記憶上に電紙間のリンクをポインタとして格納する．

（4）複数のモジュールでシステムを構成する

システムでは，電紙の永続性の管理や，表示一体型タブレットの管理など，様々な管理を行うことになる．このために，V4 では，核 OS の上に，資源管理別のモジュールを実現することによって，一つのシステムを構築する（図1）．

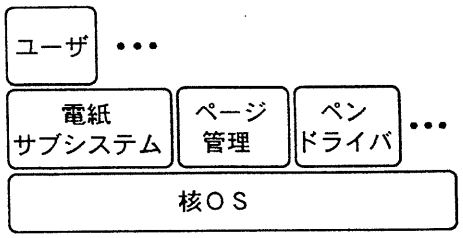

図1　システムの全体構成

3．システム全体の方針

V4 からの要求としては，次のものがある．

（1）紙の属性を OS の機能としてユーザが定義可能としたい

紙に様々な属性をユーザが定義していくことによって，OS では電紙，液晶タブレットなどの様々な資源を管理する必要がでてくる．しかし，これを単一の OS で実現した場合，OS が定める資源管理のインタフェースによっては，様々な資源管理に対応できなくなる可能性がある．

（2）ペンのレスポンスを保証したい

文字認識は，非常に計算機のパワーを必要とする．しかし，文字認識の処理中であるからといって，ペンの反応がなくなってしまうと，非常にユーザにとって使いにくいインタフェースとなってしまう．したがって，ペンのレスポンスを保証する必要がある．

これらの理由から，我々は単一の OS の中に，すべての資源管理を詰め込むのには無理があると考え，資源管理の柔軟さの面から，OS を2層に分割することにした．下の層で OS を実行するための OS を実現し，上の層で資源管理別の OS を実行させる方法をとる．また，この方法をとることによって，V4 システム全体がマルチタスク化されるため，ペンのリアルタイム性も保証することができる．

4．核 OS の設計方針

核 OS の最大の設計方針は，V4 を構成するために，必要最小限の資源管理を行うことである．そこで，次の資源を必要最小限と考えた．

（1）セグメント

V4 では，紙をセグメントとして管理している．したがって，「電紙サブシステム」，「認識系」などで，紙を共有することが必要になる．したがって，セグメントの管理は，上位層で行うことはできない．このために，核 OS ではセグメントを管理することにし，タスクがセグメントを操作できる機能を提供する．

しかし，核 OS でページまでを管理することにすると，ワンレベルストアを実現するために，核 OS 内に2次記憶装置の管理などが入ってくる．このことから，核 OS ではセグメントを管理するが，ページについては上位層で管理することにした．これによって，保護の問題が発生するが，ワンレベルストアを実現するために，ページ管理は核 OS の外で行う方針をとる．

（2）個々の資源管理をタスクとして実行する

核 OS の上で様々な資源管理を実行するために，核 OS では，プロセッサをタスクとして仮想化する．

デバイス管理をタスクが行えるようにするためには，タスクが外部割込みを受けることが必要となってくる．また，ページング処理を上位で行えるようにするためには，タスクがプロセッサのフォルトを受けることが必要である．これらのことから，核 OS では，タスクが外部／内部割込みを受けることができるようにする．

しかし，核 OS ではこれらの割込みをメッセージとして仮想化しない．このような高度な仮想化を行うとオーバヘッドが増加するし，その必要はないからである．資源管理を行うためには，タスクが割込みの通知を受け取るだけで十分である．したがって，核 OS では割込みを通知することしか行わない．

（3）デバイスの管理は行わない

核 OS でデバイスを管理してしまうと，上位層

でデバイス管理を自由に行うことができなくなってしまう．また，新しいデバイスを接続する場合に，核 OS を変更することが必要となり，保守性が低下する．このことから，核 OS ではデバイスの管理は行わず，タスクの機能で実現する．

5．核 OS の基本設計

ここでは，核 OS の基本設計について述べる．まず，タスクについて説明し，その後メモリ管理について述べる．

5．1　タスク

核 OS のタスクは，ダイナミックリンクによって，アクセス／実行していく制御の流れとなる．具体的にタスクが利用できる機能としては，次のものを設定した．これを図２に示す．また，SVC の一覧を表１に示す．

（１）仮想的な割込み

割込みの通知を受けられるようにするために，核 OS では，それぞれのタスクに仮想的な割込みベクタテーブルを用意することにした．核 OS では，タスクのコンテキストを管理し，割込みを仮想化してタスクに通知する．タスクには，仮想化されたプロセッサから割込みを受けるようなインタフェースを提供する（図３）．なお，核 OS ではタスクに対して割込みのロック／アンロックのインタフェースを提供し，ロックされているタスクに対して割込みを通知することにする．

（２）セグメントの操作

プロテクションリングを用いた保護（後述）を行うために，セグメントの属性には，保護属性，大きさといったものの他に，特権レベルが必要となってくる．したがって，セグメント生成時に特権レベルなどの属性を指定可能にする．他のセグメントの操作としては，セグメントの削除，属性の変更を用意することにした．

（３）タスクの操作

資源管理別にタスクを実行する場合には，タスク間での資源の継承などの機能は必要ない．このために，核 OS では，タスクに親子関係をもたせたりはしない．しかし，核 OS のタスクは資源を

表１　核ＯＳのＳＶＣ一覧

名　称	機　能
タスクを生成する	タスクを生成する．生成したタスク id を返す．
タスクを消去する	引数で指定されたタスク id のタスクを消去する．
タスクidを得る	この SVC を発行したタスクのタスク id を返す．
CPUを放棄する	この SVC によって，他のタスクが実行される．
タスクの状態を得る	引数で指定されたタスクの状態を返す．
セグメントを生成する	引数には，保護属性，ページテーブルアドレス 大きさなどを指定する．セグメントidを返す
セグメントを消去する	引数で指定されたセグメントを消去する．
セグメントの属性を変更する	引数で指定されたセグメントの属性を変更する．
セグメントの属性を得る	引数で指定されたセグメントの属性を返す．
割込み手続きを登録する	ベクタテーブルに割込みハンドラのアドレスを登録する．
割込みをロックする	引数で指定した割込み番号の割込みをロックする．
割込みをアンロックする	引数で指定した割込み番号の割込みをアンロックする．
仮想割込みを発生する	引数には，割込み番号，通知するタスク id を指定する．
割込みを待つ	割込み発生を待つ．引数には割込み番号を指定する．
セマフォを生成する	数にはセマフォの初期値を指定する．
Ｐ命令	引数で指定したセマフォに対して P 命令を発行する．
Ｖ命令	引数で指定したセマフォに対して V 命令を発行する．

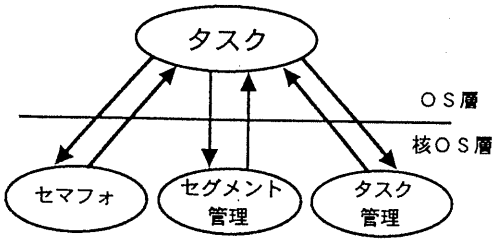

図２　タスクが利用できる機能

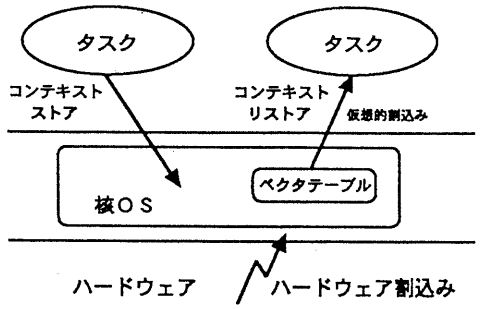

図３　仮想的な割込み

管理するため，タスクが消滅するときには，そのタスク自身に資源の終了処理を行わせるとよい．そこで，通常タスクを消去する場合には，仮想化された割込みを用いることにした．タスクはあらかじめ決められた割込み番号に終了手続きを登録しておき，タスクを消去する場合には，そのタスクに対してタスク消去の割込みを発生し，割込まれたタスクは自分自身で資源の後処理を行って終了する．

（４）セマフォ

核 OS では，デバイスの管理を行わないため，複数のタスク間での排他制御が問題となる．このために，核 OS では，セマフォを提供することにした．これによって，核 OS のタスク間の排他制御を行う．

５．２　メモリ管理

核 OS では，セグメントの管理だけを行い，ページの管理までは行わないことは前に述べた．図４にセグメントとページ管理の関係を示す．

核 OS のタスクは，それぞれが資源管理を行うことになるが，この場合には，タスク間でセグメント（紙）を共有することが必要となる．しかし，拡張された資源管理部が，他を破壊することは十分に考えられる．このことから，セグメントを共有でき，保護を考慮することが必要となる．

核 OS では，タスク間でセグメントを共有し，しかも保護を行うために，セグメントを共有できる空間と，そうでなく，タスクごとに個別のメモリ空間を与えることにした．具体的には，図５のようなメモリ空間となる．グローバルな空間に置かれたセグメントは，すべてのタスクから可視となる．V4 では，このグローバルな空間を永続的な電紙が置かれる空間として利用する．

核 OS ではセグメントの管理しか行わないため，核 OS のタスクではページを管理する．さらに核 OS がデバイスの管理を行わないことによって，主記憶のページ管理と２次記憶のページ管理をタス

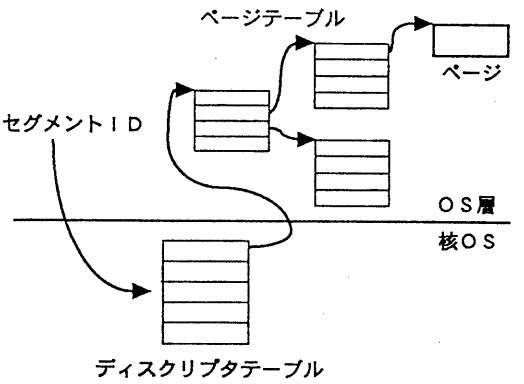

図４　セグメンテーションとページング

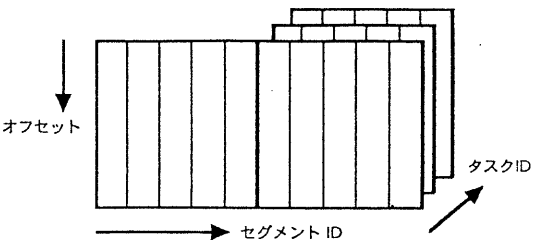

図５　メモリ空間

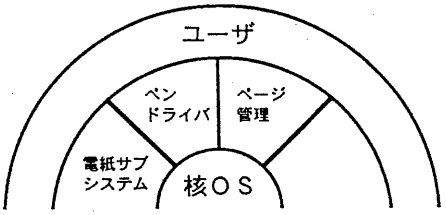

図6　プロテクションリング

クで行い，ワンレベルストアを可能とする．

さらに，「核 OS」「V4 システム」「ユーザ」という層に関して保護を行うことも重要である．したがって，核 OS では，プロテクションリングの機構を用いることで保護を行う（図6）．プロテクションリングは，セグメントの特権レベルによって，アクセスを制限するものである．これによって，従来のような，フラットなメモリ空間での特権状態／非特権状態という保護に対して，セグメントごとに複数の特権レベルを設定することができ，有効である．

6．　実行環境

核 OS および V4 のシステム記述言語としては，言語 C を用いる予定である．ダイナミックリンク／セグメンテーションに対応した言語 C の実行環境の設計[7]についてここで述べる．

言語 C コンパイラでは，コンパイル単位ごとに図7に示すようなセグメントの割当てを行う．このように，リンクテーブルは独立した一つのセグメントである．このために，リンクテーブルを動的に拡張することが可能である．また，静的なデータを一つのセグメントとして分離したことで，データを共有することが可能となっている．

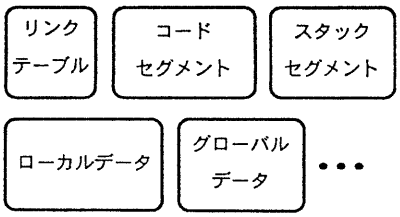

図7　セグメントの割当て

7．実現

7．1　ターゲットプロセッサ

ここでは，核 OS の実現について述べる．これまで述べた核 OS は，現在 80386 以上のプロセッサをターゲットとして実現を行っている．これは次の理由による．

（1）プロセッサがセグメンテーション，ページング，プロテクションリングをサポートしている

セグメンテーションやダイナミックリンクを実現するためには，ハードウェアでこれらをサポートしていることが望ましい．そこで，80x86 系プロセッサをネイティブモードで利用する．

（2）マシンの内部仕様が公開されている

システムの実現には，マシンの内部仕様が公開されていることが重要である．80x86 系のプロセッサを使用したマシンは内部仕様が公開されているものが多い．

また，ターゲットマシンについては，当面 PC-98 シリーズを用いることにした．これは，やはり一般に広く普及しているために実行機の入手が容易であることと，内部仕様が公開されていることによる．

7．2　開発環境

現在，我々の研究室では，68000 系プロセッサを採用したワークステーション上で，OS/omicron V2 を開発環境として用いている．核 OS の実現方

法としては，この OS/omicron 上に 80x86 系の開発環境を構築した．これは，一般に入手できるプロテクトモード用の高級言語処理系が，我々の定めたセグメンテーション／ダイナミックリンクをサポートする言語 C の実行環境とは異なるからである．

また，OS/omicron ではシステムのコードとしてフル２バイトコードを採用している．このために日本語識別子を用いたプログラミング言語を用いることができる．日本語識別子を用いることは，保守性の面で有効であると考える．

現在，この実行環境に対応した日本語識別子を用いることができる言語 C コンパイラ CAT が開発中であり，アセンブラなどのツールが実現されている．

8．おわりに

本報告では，紙の仮想化を行う OS，OS/omicron V4 を実現するための核 OS の基本設計について述べた．これからは，核 OS を完成させ，OS/omicron V4 の実現を進めていく．また，他の OS に対する有効性を確認するために，我々の研究室で開発された OMICRON V3 を実現する予定である．

9．参考文献

[1] Sigeru Motobayasi, Takasi Masuda and Nobumasa Takahasi: "THE HITAC 5020 TIME SHARING SYSTEM", 'Proc. ACM 24th, Nat. Conf. 24, pp.419-429

[2] E.I. Organick, 菊池豊彦, 佐々木彬夫：“MULTICS システム（上）（下）”，共立出版，1973

[3] 高橋延匡：“研究プロジェクト総説 OS/omicron の開発”，情報処理学会オペレーティングシステム研究会報告 39-5, 1988

[4] 早川栄一，並木美太郎，高橋延匡：“手書きインタフェースを支援する OS OS/omicron 第４版の構成”，情報処理学会第４回コンピュータシステムシンポジウム論文集，1992, pp.35-42

[5] 早川栄一，並木美太郎，高橋延匡：“紙の仮想化を行う OS/omicron 第４版における「電紙」の設計”，情報処理学会第 46 回全国大会講演論文集，1993 年 3 月

[6] Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Aavadis Trevanian and Michael Young: "Mach: A New Kernel Foundation For UNIX Development", USENIX Summer'86 pp.93-112

[7] 中村浩之，田中広幸，森永智之，早川栄一，並木美太郎，高橋延匡：“80386 プロセッサにおけるリンケージフォールト処理の一方式”，情報処理学会第 46 回全国大会講演論文集，1993 年 3 月