システムソフトウェアと
オペレーティング・システム 75－2
（1997．6．5）

マイクロカーネルLavenderの構成

芝 公仁† 佐脇 秀登† 豊岡 明† 毛利 公一† 大久保 英嗣††

†立命館大学大学院理工学研究科
††立命館大学理工学部情報学科

我々は，カーネル機能の柔軟な拡張とプロセス間の協調作業の効率化を目標として，Lavenderマイクロカーネルの実装を進めている．本論文では，Lavenderの特徴的機能であるプロセスグループとレジデントアドレス空間について述べる．これらの機能は，効率的なプロセス間の協調作業を実現するためのものであり，特徴的なメモリ管理によって実現されている．本論文では，プロセスグループ機能の評価についても述べる．

Structure and Organization of Lavender Micro Kernel

Masahito Shiba† Hideto Sawaki† Akira Toyooka† Koichi Mouri†
Eiji Okubo††

†Graduate School of Science and Engineering, Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu, Shiga 525-77, Japan

††Department of Computer Science,
Faculty of Science and Engineering, Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu, Shiga 525-77, Japan

In order to achieve the highly customizable kernel and the efficient cooperation between processes, we have been developing Lavender micro kernel. In this paper, the process group and the resident address space which are distinctive functions of Lavender micro kernel are described. These functions aim at realizing efficiently cooperative works between processes, and are implemented by the distinctive way in the memory management. Furthermore, in this paper, the performance evaluation of the process group is described.

1 はじめに

オペレーティングシステムの構成法の一つとしてマイクロカーネル方式がある. マイクロカーネルには, システムのサイズを小さくできる, 拡張性や柔軟性が高いなどの利点がある.

しかし, このようなマイクロカーネルにもいくつかの問題がある. その一つに機能の拡張に関するものが挙げられる. マイクロカーネルがシステムサーバに提供する機能が固定化され柔軟性に欠ている, あるいはマイクロカーネル内に情報が隠蔽されており, ユーザが必要な情報を参照, 変更できないようなことがある. このような場合, マイクロカーネル方式の重要な特徴である拡張性が失われてしまう.

また, プロセス間の協調作業の効率も問題となる. マイクロカーネル方式のオペレーティングシステムでは, 機能の多くを複数のシステムサーバで実現する. そのため, カーネルとシステムサーバ間, あるいは複数のシステムサーバ間での協調作業が頻繁に発生する. システムサーバがそれぞれ異なるアドレス空間に存在する場合, 協調作業時に大きなオーバヘッドが生じることになる.

現在我々が構築中であるマイクロカーネル Lavender は, これらの問題を解決するために, 階層化インタフェース, プロセスグループ機能, レジデントアドレス空間と呼ばれる機構を持っている.

階層化インタフェースとはカーネル内部の機能を階層化し, ユーザに各階層の機能を提供するものである. ユーザはこの階層化インタフェースを用いることで, 柔軟にオペレーティングシステムの機能を変更, 拡張することができる.

プロセスグループ機能は, 同一アドレス空間に複数のプロセスを配置するための機能である. この機能によりグループ化され同一アドレス空間に配置されたプロセスは, 効率のよい協調作業を行うことができる.

レジデントアドレス空間は, プロセスのグループ化を支援するための機能である. レジデントアドレス空間は, カーネルアドレス空間と同様にアドレス空間の切り替えに影響されず, 常に仮想アドレス空間上に存在する領域である. この領域に配置されたプロセスはユーザアドレス空間に配置されたすべてのプロセスからグループ化されているように見える. レジデントアドレス空間にシステムサーバを配置することで, 効率のよいサービスを実現することができる.

以下 2 章で Lavender の特徴について述べる. 3 章ではプロセスグループ機能, レジデントアドレス空間を実現するメモリ管理の方法について述べる. 続いて 4 章ではプロセスグループ機能, プロセス間手続き呼び出しについての評価を行う. 最後に 5 章で本論文のまとめを述べる.

2 Lavender マイクロカーネルの特徴

2.1 マイクロカーネル方式

Lavender は, マイクロカーネル方式で構築されているオペレーティングシステムである. Lavender は, オペレーティングシステムとして必要とされる基本的な機能を持つ Lavender マイクロカーネルと, オペレーティングシステムとしてのサービスを実現する複数のシステムサーバから構成される. Lavender ではポリシとメカニズムは分離されており, 基本的な機能のみを提供する Lavender マイクロカーネルはメカニズムを実現するためのものである. また, Lavender マイクロカーネルの提供するメカニズムを利用してポリシを実現するものがシステムサーバである.

Lavender マイクロカーネルは, 仮想アドレス空間, プロセス, スレッド, スケジューリング, プロセス間通信, 割り込みの管理のようなオペレーティングシステムとしての最小限の機能を実現する. これらの機能は, メモリマネージャ, プロセスマネージャ, スケジューリングマネージャ, プロセス間通信マネージャ, 割り込みマネージャなどの複数のマネージャにより実現される.

Lavender において, システムサーバはシステムのポリシを実現するためのユーザプロセスである. システムサーバは Lavender マイクロカーネルと協調し処理を行うことで, オペレーティングシステムとしての様々なサービスを実現する. また, ユーザは, 新たにシステムサーバを作成することで, デフォルトのポリシを置き換えることができる.

2.2 階層化インタフェース

Lavender マイクロカーネルは, ニュークリアス層, カーネル層, システム層の 3 層に階層化されている. これらの機能は以下のようになっている.

- ニュークリアス層
  直接ハードウェアを操作する機能を提供する. ユーザはこの層の機能を利用することで, ハードウェア特有の機能を利用することができる.
- カーネル層
  ニュークリアス層の機能をまとめて意味のある機能とし, オペレーティングシステムのメカニズムを実現する. また, ハードウェアの抽象化を実現している.
- システム層
  Lavender のデフォルトのポリシを実現する. ユーザはシステムサーバを作成することで, この層の提供する機能を置き換えることができる.

階層化インタフェースとは, これら階層化された各層の機能をユーザに提供するものである. マイクロカーネル方式のオペレーティングシステムでは, カーネルは必要最小限の機能をユーザに提供するが, その機能がユーザにとって必要以上に高機能であったり, 融通がきかなかったりする場合がある. また, カーネル内部に管理情報が隠蔽されており, システムサーバがこれらを利用することは困難であった. このような場合, マイクロカーネル方式のオペレーティングシステムの重要な特徴である拡張性が阻害されてしまう. Lavender マイクロカーネルの階層化インタフェースは, このような問題を解決する. ユーザは階層化インタフェースを使用することで, 処理の内容にあったシステムインタフェースを選択することができる. また, 従来はカーネル内部に隠蔽されていた情報を利用することで, より柔軟にオペレーティングシステムのポリシを変更, 拡張することが可能となる.

2.3 プロセスグループ機能

プロセスグループ機能は, 同一のアドレス空間に複数のプロセスを配置することを許す機能である. この機能によりグループ化されたプロセスは, 効率のよいプロセス間の協調作業を行うことができる.

プロセスグループ機能によりグループ化され同一アドレス空間に配置されたプロセスは, アドレス空間の切り替えを伴わず協調作業を行うことが可能である. また, グループ化されたプロセスは, 互いのメモリ資源を共有することができる. これにより, 協調作業を行うプロセス間でデータやコードを共有することが可能となる. コードを共有することで, プロセス間手続き呼び出しを自プロセスの手続き呼び出しと同様に行うことができる. この場合, メッセージ通信を使用したり, カーネルを介す必要がないため非常に効率がよい.

2.4 レジデントアドレス空間

Lavender のアドレス空間は, 1 つのカーネルアドレス空間, 複数のユーザアドレス空間, 1 つのレジデントアドレス空間から構成される. カーネルアドレス空間には, 保護されたカーネルのコードとデータが配置される. ユーザアドレス空間には, 複数のユーザプロセスが配置される. レジデントアドレス空間には, ユーザアドレス空間と同様に複数のユーザプロセスが配置される. ただし, レジデントアドレス空間はアドレス空間の切り替えに影響されず, 常に仮想アドレス空間上に存在する. そのため, レジデントアドレス空間に配置されたプロセスは, ユーザアドレス空間に配置されたすべてのプロセスからグループ化されているように見える. このレジデントアドレス空間に他のプロセスと頻繁に協調作業を行うシステムサーバのようなプロセスを配置することで, アドレス空間の切り替えを軽減させることができる.

3 メモリ管理

3.1 プロセスグループ機能

マイクロカーネル方式のオペレーティングシステムでは, 密接なプロセス間の協調作業が必要とされる. しかし, 協調作業を行うプロセスが異なるアドレス空間に配置されていると, 協調作業時に無視できないオーバヘッドが発生する.

Lavender のプロセスグループ機能は, 同一のアドレス空間に複数のプロセスを配置するための機能である. 同一のアドレス空間に配置されたプロセスは, アドレス空間の切り替えを行うことなく協調作業を行うことができる. また, グループ化されたプロセスは, オーバヘッドを伴わないプロセス間手続き呼び出しを行うことができる. そのため, グループ化されたプロセスは効率のよい協調作業を行うことが可能となる.

Lavender において, プロセスは複数のセグメントから構成される. セグメントとは同一の保護属性を持つ連続したアドレス領域のことである. セグメントにはテキスト, データ, スタックの 3 種類があり, 各セグメントは SDT 構造体により管理される. SDT 構造体は, セグメントの配置されるリニアアドレス, セグメントの大きさ, セグメントの保護属性などの情報を持っている. プロセスのアドレス空間への配置は, セグメントを単位として行われる.

各プロセスは, SDT 構造体が登録されるセグメントディスクリプタテーブルを持っている. プロセスは, このテーブルに登録されている SDT 構造体が示すセグメントを使用することができる. 各プロセスは同一グループの他のプロセスが持つ SDT 構造体を, 自プロセスのセグメントディスクリプタテーブルに登録することができる. この機能により, プロセスは他のプロセスのセグメントを自プロセスの所有するセグメントであるかのように使用することができる. つまり, グループ化されたプロセスはセグメントの共有が可能となる.

例えば, データセグメントを共有することで, 簡単に共有メモリを利用することができる. また, コードセグメントを共有し, コードを共有することも可能である. コードを共有することにより, 他のプロセスの手続きを自プロセスの手続きであるかのように呼び出すことができる. このようにして実現されるプロセス間手続き呼び出しは, オーバヘッドがなく効率がよい.

従来のプロセス間手続き呼び出しでは, メッセージ通信, アドレス空間の切り替え, CPU コンテキスト切り替えによるオーバヘッドが問題となっている. しかし, Lavender では, 同一アドレス空間内でセグメントを共有し, そのセグメント内のコードを直接実行することが可能である. そのため, プロセス間手続き呼び出しにオーバヘッドが発生せず, 効率的にプロセス間の協調作業を行なうことができる.

3.2 レジデントアドレス空間

マイクロカーネル方式のオペレーティングシステムでは, オペレーティングシステムとしての機能の多くを複数のシステムサーバで実現する. これらのシステムサーバは, 互いに協調することでサービスを行う. 例えば, プロセスを生成する場合, プロセスサーバ, ファイルサーバ, メモリサーバ, スケジューリングサーバの 4 つシステムサーバが機能することになる. これらのサーバが互いに異なるアドレス空間に配置されている場合, アドレス空間の切り替えが頻繁に発生する. この問題は, プロセスグループ機能を利用しシステムサーバをグループ化することで解決できる. しかし, システムサーバがグループ化されていてもサービスを要求するプロセスが異なるアドレス空間にある場合, 効率のよいサービスを行うことができない.

Lavender では, 仮想アドレス空間内にレジデントアドレス空間と呼ばれる領域を持つことでこのような問題を解決している. レジデントアドレス空間は, アドレス空間の切り替えに影響されず, 常に仮想アドレス空間上に存在する領域である. レジデントアドレス空間には, 複数のユーザプロセスを配置することができる. レジデントアドレス空間に配置されたプロセスは, 常に仮想アドレス空間上に存在する. そのため, レジデントアドレス空間に配置されたプロセスは, ユーザアドレス空間に配置されたすべてのプロセスからグループ化されているように見える. 複数のグループのプロセスと協調作業を行うシステムサーバのようなプロセスをレジデントアドレス空間に配置することで, アドレス空間の切り替えを軽減させることができる.

Lavender には複数のアドレス空間があり, それらが切り替えられて使用される. 複数のアドレス空間を使用するために, MMU のページング機能を利用している. すなわち, 使用するアドレス空間の数だけのページテーブルを用意し, それらを切り替えてアドレス空間を切り替えている. しかし, 各アドレス空間においてカーネルアドレス空間, レジデン

トアドレス空間は同じでなければならない．すなわち，各ページテーブルで，カーネルアドレス空間，レジデントアドレス空間にあたる部分は同じでなければならない．現在 Lavender の実装を行っている i486 では，ページテーブルは 2 段階のものとなっている．この性質を利用すると，ページテーブルの一部を共有することができる．ページテーブルの一部を共有することにより，共有部分の管理するアドレス領域はすべてのアドレス空間で同一のものとなる．また，カーネルアドレス空間，レジデントアドレス空間内でページの保護属性等の変更があった場合，その変更は 1 つのページテーブルで変更を行うだけでよい．この方法を用いると，ページテーブルの変更は常に 1 回でよいことになる．ページテーブルの変更に要する時間が，アドレス空間の数によらず一定であることは，時間的制約があるようなシステムにおいては重要なことである [1].

4 評価

4.1 プロセスグループ機能

プロセスグループ機能によるプロセス間の協調作業の効率化を調べるために，プロセスがグループ化されている場合，いない場合の実行効率を比較した．

2 つのプロセスが協調作業を行うとき，各プロセスが異なるアドレス空間にある場合，アドレス空間の切り替えが発生する．また，アドレス空間の切り替えは，TLB キャッシュのフラッシュを伴う．TLB エントリは仮想アドレスと物理アドレスの変換を効率よく行うためのキャッシュである．この TLB キャッシュのフラッシュを行うと，アドレス変換に逐一ページテーブルの参照が行われ，CPU の実行効率が低下する．一方，協調作業を行うプロセスをグループ化し同一アドレス空間に配置すると，アドレス空間の切り替えが発生しない．この場合，TLB キャッシュのフラッシュは行われない．そのため，プロセスを切り替えても CPU の実行効率は低下せず，各プロセスは効率よく処理を行うことができる．

例えば，2 つのプロセスが協調作業を行うとする．これらのプロセスは 16Kbyte，すなわち 4 ページ分の大きさで，交互にディスパッチされ，I/O 待ちはないとする．これらのプロセスが，グループ化されている場合といない場合に，一定の処理を行うのに要した時間の比を表 1に示す．計測には Pentium/200MHz を使用した．協調作業が少ない場合，1 つのプロセスに対する 1 回の CPU の割り当て時間が十分長いため TLB キャッシュのフラッシュの影響をほとんど無視できる．そのため，グループ化されている場合といない場合の差は，アドレス空間の切り替えの有無であり 1 割程度となっている．一方，協調作業が多い場合はグループ化により 5 割近い差が現れている．プロセスが協調作業を行う場合，プロセスは CPU を割り当てられた時間を使いきる前に他のプロセスに CPU を譲ってしまう．そのため，TLB キャッシュがフラッシュされるまでの時間が短くなり，TLB キャッシュが有効に働くことができなくなる．しかし，プロセスがグループ化されている場合，アドレス空間を切り替えるオーバヘッドがなくなる．また，プロセスの切り替え時に TLB キャッシュがフラッシュされないため，TLB が常に有効に働くことができる．

表 1 所要時間比

	グループ化	
協調作業	していない	している
少	1.00	0.92
多	1.00	0.54

今回の実験では，協調作業が多い場合と少ない場合を比較した．しかし，協調作業時のプロセスの切り替えのオーバヘッドは，その頻度ではなくグループ化しているかいないかに依存する．そのため，協調作業のオーバヘッドそのものは，グループ化しているかいないかで 5 割以上の差があると考えられる．このことから，協調作業を行うプロセスをグループ化することは協調作業の効率化のために有効であるといえる．

4.2 プロセス間手続き呼び出し

ユーザアドレス空間に配置されたプロセスが，レジデントアドレス空間に配置されたプロセス内の手続きを呼び出したときの所要時間を計測した．その結果を他の環境でのプロセス間手続き呼び出しと比

較したものを表 2に示す [2].

メッセージ通信を用いてプロセス間手続き呼び出しを行う場合, 次の 2 つのオーバヘッドが生じる.

(1) スレッドの切り替えが発生し, CPU コンテキストの切り替えが必要となる.

(2) アドレス空間の切り替えが発生する.

また, 呼び出された手続きの実行は呼び出し側と異なるスレッドが行うため, 手続きの完了はスケジューリングに左右される. また, メッセージ通信自体もコストの大きい処理である.

Spring は効率のよいプロセス間手続き呼び出しを実現している. SPARCstation 2 上の Spring において, ユーザプロセスは最小限の引数を持つ他のプロセスの手続きを $11\mu s$ のオーバヘッドで呼び出す. これは, スレッドを切り替えずにプロセス間手続き呼び出しを行い, CPU コンテキスト切り替えのオーバヘッドをなくしているためである.

さらに Lavender では, プロセスグループ機能とセグメントの共有を利用することで, スレッドの切り替えだけでなくアドレス空間の切り替えによるオーバヘッドもなくすことができる. アドレス空間の切り替えは TLB キャッシュのフラッシュを伴い, 切り替え後の実効効率が低下するという問題がある. しかし, Lavender では TLB キャッシュをフラッシュする必要がないため, 呼び出した手続きを効率よく実行できる. 実際, i486DX2/66MHz 上の Lavender において, ユーザアドレス空間のプロセスはレジデントアドレス空間のプロセス内の手続きの呼び出し及び復帰を $0.75\mu s$ で行う.

このようにして実現するプロセス間手続き呼び出しは, オーバヘッドのない効率のよいものであるがいくつかの問題がある. まず, グループ化されていなければならないため利用できる場合が限られる. 異なるグループのプロセス間でも物理メモリレベルでのコードの共有は可能であるため, このような問題は解決できると考えられる. また, プロセス間でのデータの受け渡しも問題となる. 特に, データが call-by-reference の場合問題となる. これを解決するために, 手続き呼び出しにカーネルを介す方法も考えられる. しかし, その場合オーバヘッドがないという利点が失われてしまう. また, データの受け渡しに制限を設けるという方法も考えられるが, 使用方法が限られるという問題がある. これらの問題については今後の課題である.

表 2 プロセス間手続き呼び出しの所要時間

	Sockets	Spring	Lavender
raw time	$850\mu s$	$11\mu s$	$0.75\mu s$
dhrystone	27	27	31
scaled time	22950	297	23

5 おわりに

本論文では, Lavender の全体構成, 及び階層化インタフェース, プロセスグループ機能, レジデントアドレス空間について述べた. 階層化インタフェースは, ユーザが柔軟にオペレーティングシステムの機能を変更, 拡張することを可能とする. また, プロセスグループ機能, レジデントアドレス空間により, プロセスは効率のよい協調作業を行うことができる. これは, オペレーティングシステムとしての機能の多くを複数のシステムサーバで実現するマイクロカーネル方式のオペレーティングシステムにとって重要なことである.

また, プロセスグループ機能, プロセス間手続き呼び出しについての評価も行った. これらの機構を利用することで, プロセス間の協調作業時に生じるオーバヘッドの軽減が可能であることを確認できた. しかし, 現在実装されているプロセス間手続き呼び出しにはいくつかの問題があり, これらの解決は今後の課題である.

参考文献

[1] 河野通宗, 前沢敏行, 安西祐一郎: "リアルタイム *OS* における単一仮想空間のマッピング手法", 情報処理学会研究会報告 96–OS–73, pp. 1–6 (1995).

[2] Panos Kougiouris and Graham Hamilton: *"The Spring nucleus: A microkernel for objects"* , Proceedings 1993 Summer USENIX Conference, pp. 147-160 (1993).