システムソフトウェアと
オペレーティング・システム 79－5
（1998．8．6）

マイクロカーネル Lavender 上への ネットワークシステムサーバの構築

森田 健司† 佐脇 秀登† 芝 公仁† 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

マイクロカーネル Lavender は，ユーザカスタマイズ可能なカーネルとして構築されている．Lavender は，ポリシとメカニズムの分離，階層化インタフェース，クロスアドレススペースコールのオーバヘッドの軽減などの特徴を持つ．本論文では，Lavender におけるネットワーク機構と，その応用について述べる．Lavender では，ネットワークインタフェースの動的な追加及び変更を可能とするために，デバイスドライバおよびプロトコル処理部をシステムサーバプロセスや LKM(Loadable Kernel Module) として実現している．

An Implementation of Network System Server on Lavender Micro Kernel

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

†Graduate School of Science and Engineering, Ritsumeikan University
1-1-1 Noji-Higashi, Kusatsu, Shiga 525-8577, Japan

††Department of Computer Science,
Faculty of Science and Engineering, Ritsumeikan University
1-1-1 Noji-Higashi, Kusatsu, Shiga 525-8577, Japan

Lavender micro kernel is designed as a user customizable kernel. Lavender has following features: separation of policy and mechanism, layered interface and decrease of overhead in cross-address-space call. In this paper, the structure of network system in Lavender micro kernel and its application are described. In Lavender micro kernel, device driver and protocol stack are implemented as system process or LKM(Loadable Kernel Module) in order to dynamically add and remove network interface.

1 はじめに

近年, ネットワーク機器の低価格化やインターネット技術の発展により分散環境を構築することが容易となり, OS にも簡単にネットワークを利用できる機構が必要とされている. また, モバイルコンピューティングの急速な普及などにより, PC カードなど各種外付け機器を使用する機会が増え, ハードウェアの動的な変更への対応が OS に求められている.

しかし, UNIX など従来の OS におけるネットワーク機構はカーネル内で実現されているため, ユーザにとって必要なサービスが実現されていなかったり, 逆に不要なサービスを含んでいたりした. また, これらのサービスを動的に追加, 削除, 変更をすることは困難であった.

我々はこのような状況に対応するため, ユーザカスタマイズ可能なマイクロカーネル Lavender [1][2][3][4][5][6] を構築している. マイクロカーネル方式の OS では, システムサーバを置き換えることで機能の変更, 拡張が容易に行える. これにより, 従来のカーネルの持つ問題点を解決することができる. Lavender では, マイクロカーネルの持つ柔軟性と拡張性の高さを活用し, 個々のユーザが必要とする機能を動的に追加し, 提供することを目的としている. 本論文では, 特に Lavender 上へのネットワークシステムサーバの構築について述べる.

我々が現在構築している Lavender におけるネットワーク機構では, デバイスの動的な交換を可能をするためにデバイスドライバの Loadable Kernel Module(LKM) とシステムサーバによる実装をサポートしている. 時間制約が厳しいデバイスに関しては LKM を用いて実装を行い, 拡張性を求められるデバイスに関してはシステムサーバとして実装する.

また, Lavender の提供する IPC 機構を使用してカーネルや他のシステムサーバとの協調作業を行う. IPC 機構を使用することにより, ユーザーは特に意識せずに, 他ノード上のデバイスドライバやシステムサーバを自ノードのものと同様に扱うことが可能となる.

本論文では 2 章で Lavender の構成と特徴を述べ, 3 章でネットワークサーバの実装について述べる. また, 4 章でネットワークサーバの利用について述べる.

2 Lavender の特徴と構成

2.1 Lavender マイクロカーネルの特徴

Lavender マイクロカーネルは, カーネルアドレス空間に配置され, カーネルモードで動作する. ユーザプロセス, デバイスドライバ, システムサーバはユーザアドレス空間に配置される.

(1) 階層化インタフェース

階層化インタフェースは, カーネル内部の機能を階層化し, システムサーバやユーザプロセスに各層の機能を提供するものである. カーネル内部はニュークリアス層, カーネル層, システム層の 3 層に分割され, それぞれの階層で異なった粒度のシステムインタフェースを提供している.

- **ニュークリアス層**
  ハードウェアを直接操作する機能を提供する層である. この層を利用することで, ハードウェアに依存する小さな粒度の操作が可能である.

- **カーネル層**
  ハードウェア独立の実現と, ポリシを含まない基本機能を提供する層である. この層では, ハードウェアを抽象化したインタフェースを提供する. カーネル層を使ってプログラムされたシステムサーバなどは, ハードウェアに依存しないので, 移植が容易になる.

- **システム層**
  カーネル層の提供する基本機能を使用して構成され, ポリシを提供する層である. この層では, OS の動作に必要な最低限のポリシが提供される.

階層化インタフェースを用いることで, 従来のマイクロカーネルでは, カーネル内部に隠蔽されていた管理情報を, システムサーバが利用できる. また, 目的に応じた粒度の手続きを階層から選ぶことで, より柔軟な OS のカスタマイズが可能となる.

(2) ユーザカスタマイズ可能なカーネル

従来, カーネルの提供する機能を変更する場合は, カーネルのソースコードを変更し, 再構築する必要があった. Lavender では, ポリシとメカニズムを分離することで, カーネルに手を加えずに OS をカスタマイズすることを可能としている.

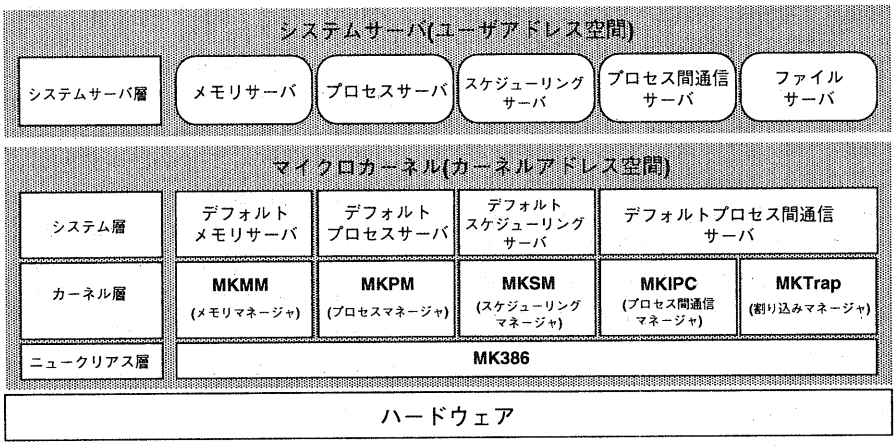

図 1　Lavender の構成

(3) ポリシとメカニズムの分離

Lavender では, メカニズム部分をカーネルで, ポリシ部分をシステムサーバで実現している. ここでポリシとは処理の手順や方法を決定することであり, メカニズムとはポリシの決定に基づき実際に処理を行う手続きをいう.

カーネルは, スレッドのコンテキストスイッチや主記憶の管理, 割り込みの制御, デフォルトのポリシなど, OS として必要な最小限の機能を実現する.

システムサーバは, スケジューリングアルゴリズムや, 主記憶の管理手法などを実現する. このため, システムサーバ部分を変更することで, OS としてのポリシを変更, 拡張することが可能である.

また, メカニズム部分をカーネルで実現することにより, ハードウェアに依存する部分はカーネル内部に局所化されるので, システムサーバはハードウェアと独立に構成することができる.

(4) アドレス空間切り替えに伴うオーバヘッドの軽減

マイクロカーネル方式の OS では, システムサーバ間の頻繁なメッセージ通信によるオーバヘッドが問題となる. Lavender では, レジデントアドレス空間とプロセスグループ機能により, このオーバヘッドを減少させている.

レジデントアドレス空間は, アドレス空間の切り替えが発生しない, ユーザアドレス空間内の領域である. レジデントアドレス空間に複数のシステムサーバを配置することで, システムサーバ間のメッセージ通信や, 処理移行時のコンテキストスイッチにおけるオーバヘッドを軽減できる.

プロセスグループ機能は, 同一アドレス空間上に複数のプロセスをひとつのグループとして配置する機能である. 同一プロセスグループに属するプロセス間での協調作業では, レジデントアドレス空間と同様の効率化が実現できる.

2.2　ネームサーバ

ネームサーバは, 内部にデータベースを持ち, ユーザの使用する情報とシステム内の管理情報の対応を管理する. ネームサーバはシステムサーバプロセスとして動作し, ネームサーバ自身もプロセス間通信を用いて通信する. ネームサーバはシステムサーバやユーザプロセスの依頼によって情報の対応の登録, 削除, 変更を行い, 問い合わせに対して, 対応する情報を返す働きをもつ.

2.3　デバイスドライバの構成

Lavender のデバイスドライバには, カーネルモードで動作するカーネルモードドライバと, ユーザモードで動作するユーザモードドライバの 2 種類がある. どちらも動的な追加及び削除が可能であり, カーネルのソースを変更及び再構築する必要はない. また, デバイスドライバは, プロセス間通信を用いて, カーネルやシステムサーバとの協調作業を実現する. このため, 他ノードのデバイスドライバを, 自ノードのデバ

イスドライバと同様に使用することができる．これら2種類のデバイスドライバの特徴を，それぞれ以下で述べる．

- **カーネルモードドライバ**
  時間制約のあるI/O処理を必要とするデバイスや，割り込み禁止区間のあるデバイスなどは，カーネルモードドライバとして実現する．デバイスドライバはLKM(Loadable Kernel Module)として実装され，カーネルアドレス空間へ動的にロードされる．

- **ユーザモードドライバ**
  ユーザモードで動作するデバイスドライバは，特定の物理メモリアドレス，I/Oポートへのアクセス権限を持った，システムサーバプロセスとして実現される．

3 ネットワークサーバ

3.1 ネットワークデバイスドライバ

Lavenderでは，デバイスドライバはLKMとユーザモードドライバがサポートされている．どちらも動的な追加および削除が可能であり，カーネルのソースの変更および再構築の必要はない．以下ではネットワークデバイスドライバの各方式における実装について述べる．

3.1.1 LKMによる実装

LKMはカーネル内に動的に組み込まれるモジュールであり，組み込まれた後は静的に結合されたデバイスドライバと同様の動作を行う．この場合，カーネルモードでデバイスドライバが動作するため，モードの切り替えが起こらず，高速な通信や時間制約のある通信を行うのに適していると考えられる．

しかし，カーネル内部のテーブルを外部から書き換えることによって組み込まれるため，動的な追加，削除を行う際に障害が起こる可能性がある．また，ドライバの機能を利用するためには，既に固定されたインタフェースを持つシステムコールに頼ることになり，デバイスドライバとユーザプロセスとの柔軟な連携をとったり，ハードウェアの特殊な機能を扱うことが難しくなるといった問題点がある．

3.1.2 ユーザモードドライバによる実装

ユーザモードで動作するデバイスドライバは，特定の物理メモリアドレス，I/Oポートへのアクセス権限を持った，システムサーバプロセスとして実現される．デバイスドライバとユーザプロセス間のインタフェースを自由に記述可能であるため，ユーザプロセスからデバイスドライバの機能を十分に利用することができる．また，カーネルに直接手を加えずに組み込むため，ドライバの追加，削除時の障害が起こりにくい．

しかし，ユーザモードで動作することによって，モードの切り替えが多くなり，高速な通信には向かないと思われる．

3.1.3 Lavenderにおける実装

今回の実装では，ネットワークドライバをシステムサーバとして実装する．ネットワークインタフェースの組み込み，削除の手順を以下に述べる．

インタフェースの組み込み

(1) ネットワークドライバは，生成されると，まずカーネルや外部サーバとの通信用ポートを取得する．

(2) 次に，ドライバは，動作に必要な資源をカーネルに通知する．

(3) カーネルは，資源に競合がなければ，割り込み処理手続きテーブルにドライバを登録し，デバイスドライバの動作に必要なメモリ，I/Oポートの参照，変更許可を与える．

(4) ドライバは，ネームサーバと通信を行い，自身を有効なドライバとして登録させる．また，ネームサーバの上位プロトコルサーバの登録を内部バッファにコピーする．後，定期的にリフレッシュする．

(5) ドライバは，ハードウェアの初期化手続きを行い，呼び出されるまで実行可能状態に移る．

インタフェースの削除

(1) ドライバは，削除要求を受け取ると，現在残っているセッションを終了させる．

(2) ドライバは, ネームサーバと通信を行い, ドライバ自身の登録された情報を削除させる.

(3) ドライバは, カーネルに終端通知を行う.

(4) カーネルは, 割り込み手続きテーブルからドライバの登録を削除する.

(5) ドライバは, ハードウェアの終端手続きを行う.

(6) カーネルは, 使用していたポートを解放し, メモリ, I/O ポートの参照, 変更許可を破棄する.

3.2 プロトコルサーバ

ネットワークプロトコルを扱うサーバの実装方式は, Lites[7] のように Unix サーバなどのシステムサーバ内に内蔵してしまう方式 (図 2a) とそれ自身をシステムサーバとして独立させる方式 (図 2b) が考えられる. システムサーバ内蔵方式は, サーバ間の通信を減らすことができ, 高速性を保つことができる. しかし, プロトコルの変更があれば Unix サーバを変更する必要がある. Unix サーバは, システムサーバではあるが, Unix システムを提供しているために非常に大きいものとなっている. このようなサーバに対する変更, 再構築はカーネルのそれと同等の労力が必要であると考えられる. よって, プロトコルスタックは, それ自身が独立したシステムサーバとするのがよいと考えられる.

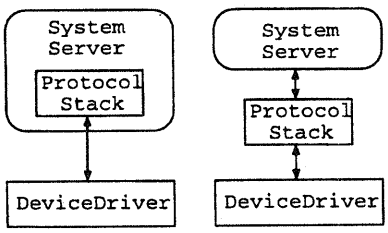

図 2 プロトコルスタックの配置

4 ネットワーク通信手順と応用

Lavender のネットワーク通信では, 基本的にポートを用いた通信を行う. 手順を以下で述べる.

(1) ユーザプロセスは, ローカルポートにメッセージを送る.

(2) プロトコルサーバは, そのメッセージを受け取り, ネームサーバに問い合わせを行う.

(3) ネームサーバは, ローカルポートとネットワークポート識別子を変換し, プロトコルサーバに返す.

(4) プロトコルサーバは, 得られた識別子から, ネットワークインタフェースを選択し, 適切なヘッダを与えたデータグラムをデバイスドライバに渡し送信する.

(5) 相手側のプロトコルサーバは, ネームサーバと通信し, ネットワークポート識別子をローカルポートへと変換し, メッセージとして送る.

このような構成にすることによって, ユーザプロセス側からは自コンピュータ上のカーネルもしくはユーザプロセスと通信をしているのか, ネットワーク上の他のコンピュータのそれと通信しているのかを意識せずに通信することができる.

他のコンピュータ上にあるプロセスとの通信をこのように行うことで, 従来の実装では難しかった, さまざまな応用が考えられる.

4.1 ネットワークサービスの実装

従来からある NFS, DNS などのネットワークサービスの構築が, それぞれ新たな処理機構を構築する必要なく, システムサーバの組合せのみで可能となる. 例えば, NFS を実装するためには, ネットワークサーバ, ファイルサーバ, ネームサーバを単純に組合せるだけでよい.

4.2 容易な新プロトコルへの対応

現在, 新たに提案されているモバイル環境に対応した通信プロトコルなどは, 現在あるプロトコルスタックのインタフェースを変更することでそのまま利用可能とすることができる. 従来の実装では, インタフェースの変更はカーネル内部の変更となり, 多大な労力が必要となる. しかし, 各処理単位ごとに独立したインタフェースを持つシステムサーバとすることで, その部分のみの変更をおこなうだけでさまざまな形式での実装が可能であり, その中から最適な実装を選ぶことができる.

また, 分散共有メモリの実装においては, プロトコルスタックを経由しない通信をおこなうことでメモリ

間のコピーを減らし, 通信コストを減少させる実装がある. このような実装もカーネルに手を加えることなく容易に行える.

4.3 複数ノード間のシステムサーバの協調

ユーザプロセスおよび他のシステムサーバは, プロセス間通信において, ネットワーク位置透過性を持っているため, システムサーバ同士の協調作業を行う際に, ノードを気にせず作業を行うことが可能となる. これによって, あるネットワークの複数のノードを, 対称構造を持つ一つのOSとして扱うことも可能となる.

5 おわりに

本論文では, Lavenderのネットワーク機構の構成について述べた.

ネットワークドライバについては, システムサーバ, LKMを使い分けて柔軟性, 高速性を重視した設計, 実装を進めている. また, ネットワークサーバを利用して遠隔IPCを行うことにより, ユーザがネットワークを意識せずに他のコンピュータを扱える機構を設計している.

現在, カーネル内部に組み込まれていたプロトコルサーバとデバイスドライバの一部が, カーネルから分離され, ユーザプロセスとして動作している. 継続して, デバイスドライバ全体をユーザプロセス化し, その性能の検証と, プロトコルサーバによるネットワーク透過機構の実装を行う予定である.

参考文献

[1] 芝 公仁, 佐脇 秀登, 豊岡 明, 毛利 公一, 大久保 英嗣: **マイクロカーネル *Lavender* の構成**, 情報処理学会研究報告 97-OS-75, 情報処理学会 (1997).

[2] 毛利 公一, 佐脇 秀登, 芝 公仁, 豊岡 明, 大久保 英嗣: **マイクロカーネル *Lavender* におけるスケジューラの構成**, 情報処理学会研究報告 97-OS-76, 情報処理学会 (1997).

[3] 豊岡 明, 佐脇 秀登, 芝 公仁, 毛利 公一, 大久保 英嗣: **マイクロカーネル *Lavender* における *IPC* 機構とデバイスドライバの構成**, 情報処理学会研究報告 97-OS-76, 情報処理学会 (1997).

[4] 毛利 公一, 佐脇 秀登, 芝 公仁, 豊岡 明, 大久保 英嗣: **マイクロカーネル *Lavender* における多段階スケジューリング機構**, 第55回 (平成9年後期) 全国大会講演論文集 (1997).

[5] 佐脇 秀登, 芝 公仁, 豊岡 明, 毛利 公一, 大久保 英嗣: **マイクロカーネル *Lavender* におけるプロセス管理方式**, 第55回 (平成9年後期) 全国大会講演論文集 (1997).

[6] 毛利 公一, 山田 博士, 斎藤 彰一, 中村 素典, 大久保 英嗣: **マイクロカーネル *Lavender* における階層化インタフェース**, 情報処理学会研究報告 95-OS-70, 情報処理学会 (1995).

[7] Johannes Helander: *"Unix under Mach: the Lites Server"*, Helsinki University of Technology, Master's Thesis (1994).