システムソフトウェアと
オペレーティング・システム 76－9
（1997．8．21）

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

豊岡 明† 佐脇 秀登† 芝 公仁† 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IPC(Inter Process Communication)機構と，デバイスドライバの構成について述べる．

Lavenderでは，コピーの発生するIPCを，共有メモリを使用することで実行時に効率化している．また，デバイスドライバをシステムサーバプロセスやLKM(Loadable Kernel Module)として実現することで，動的な追加及び削除が可能となっている．

IPC Mechanism and Device Driver in Lavender Micro Kernel

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

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

††Department of Computer Science,
Faculty of Science and Engineering, Ritsumeikan University
1-1-1 Noji-Higashi, Kusatsu, Shiga 525-77, 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 IPC mechanism and the device driver in Lavender micro kernel are described.

In Lavender micro kernel, the number of message copies for IPC is decreased at runtime by utilizing the shared memory. Furthermore, it is possible to dynamically add and remove device driver by implementing those drivers as system process or LKM(Loadable Kernel Module).

1 はじめに

従来のOS(Operating System)におけるカーネルでは, ユーザへのサービスをカーネル内で実現しているため, 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追加, 削除, 変更することは困難であった. 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.

モーバイルコンピューティングの急速な普及などにより, PCカードや各種外付け機器など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. これに対応するためには, デバイスドライバや, 各種システムサーバの動的な追加, 削除が可能でなくてはならない. また, 分散環境では, 他ノードの資源を自ノードの資源と同様に扱える必要がある.

このような状況を想定し, 我々はユーザカスタマイズ可能なマイクロカーネルLavenderを構築している. 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におけるIPC機構とデバイスドライバの構成について述べる.

我々が現在構築しているLavenderにおけるIPC機構では, コピーによるIPCのオーバヘッドを軽減するため, 共有メモリにより実行時に効率化を行う. これは, IPCライブラリとIPCサーバの協調により実現される.

デバイスドライバは, Lavenderの提供するIPC機構を使用することで, カーネルやシステムサーバと協調作業を行う. この構造により, 他ノード上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可能となる. また, 外部の物理的デバイスと入出力を行うシステムサーバとして, ユーザモードドライバを用意しており, 動的な追加及び削除が可能である. さらに, 時間制約が厳しかったり, 割り込み禁止区間を必要とするデバイスについては, カーネルモードで動作するカーネルモードドライバとして実現される.

本論文では2章でLavenderの構成と特徴を述べ, 3章でIPC機構について述べる. また, 4章でデバイスドライバの構成について述べる.

2 Lavenderの構成と特徴

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. また, OSとして必要な機能を実現するために, 複数のシステムサーバがLavenderと協調して動作する. 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.

マイクロカーネルとシステムサーバの協調作業を効率化するために, Lavenderは以下のような特徴を持つ.

(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.

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.

- **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.

(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．

(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，主記憶の管理技法などを実現する．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 [1].

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．

3 IPC 機構

Lavender ではポートを基本とした IPC と，共有メモリを使用した IPC を提供する．IPC はユーザプロセス同士の通信の他，カーネルとシステムサーバ間，カーネルとデバイスドライバ間でも使用される．以下でこれら 2 種類の IPC について，その概要を述べる．

3.1 共有メモリによる IPC

Lavender では，カーネルとユーザプロセス間，ユーザプロセス同士の間で共有メモリを持つことができる．共有メモリ内に直接メッセージを用意することで，コピーの発生しない通信を行うことができる．以下に，共有メモリによる IPC の手順を示す．

(1) 通信を行うプロセス間で，共有メモリを確保する．

(2) 送信側プロセスは，共有メモリ領域内にメッセージを用意し，メッセージ送信のシステムコールを発行する．

(3) メッセージ送信の要求を受けた IPC サーバは，メッセージの到着を受信側プロセスに通知する．

(4) メッセージの到着通知を受け取った受信側プロセスは，共有メモリ内のメッセージを受信する．

(5) 共有メモリを解放し，通信を終える．

3.2 ポートによる IPC

Lavender の IPC 機構が用意しているインタフェースには，ポートの割り当てと解放，メッセージの送受信，メッセージに関する情報の取得がある．プロセスはこれらのインタフェースを用いて，通信を行う．以下で，IPC 機構における各構成要素について述べる．

- **ポート**
  ポートは IPC の際に通信相手を特定するためのものである．通信相手は全てポート番号のみで区別されるので，通信相手がカーネルやシステムサーバ，他のユーザプロセスなど異なった場合でも，それを意識する必要はない．

  ポート番号は，プロセスにポートを割り当てる際に，IPC サーバによって内容を設定される．またポート名は，ユーザによって設定される．

- **メッセージ**
  Lavender におけるメッセージは，任意の大きさのデータのブロックである．

- **IPC サーバ**
  IPC サーバは，ポートの割り当て，解放，メッセージの送受信，メッセージやポートの状態取得などの基本的なインタフェースを提供する．

- ネームサーバ
  IPC で使用されるポート番号とポート名の対応は, ネームサーバによって管理される. ネームサーバはシステムサーバとして動作し, ネームサーバと IPC サーバも IPC を使用して通信する.

  ネームサーバは, 内部にデータベースを持ち, ポート名とポート番号の対応を管理する. ネームサーバは, IPC サーバの依頼によって, ポート番号とポート名の対応を登録あるいは削除する. またユーザプロセスは, 通信時に必要になるポート番号を, ポート名を指定してネームサーバに問い合わせる.

次に, ポートによる通信の手順を以下に示す.

(1) 受信側プロセスは IPC サーバに, ポート名を指定してポートの割り当て要求を出す.

(2) IPC サーバは, ポート名と割り当てたポート番号をネームサーバに登録し, ポート番号を受信側プロセスに与える.

(3) 送信側プロセスがメッセージを送信しようとする場合, 受信側プロセスのポート番号をネームサーバに問い合わせる. ポート番号の問い合わせにはポート名を指定する.

(4) ネームサーバはデータベースを検索し, ポート番号を送信側プロセスに返す.

(5) メッセージを送信する場合は, 受信側プロセスのポート番号, メッセージ, サイズを指定して, メッセージ送信のシステムコールを発行する.

(6) IPC サーバがメッセージ送信の要求を受け取ると, 送信するメッセージを IPC サーバ内のバッファにコピーする. 続いて, 受信側プロセスに対してメッセージの到着通知を行う.

(7) 受信側プロセスはメッセージの到着通知を受け取る. メッセージの受信を行う場合は, ポート番号を指定してメッセージ受信のシステムコールを発行する.

(8) IPC サーバがメッセージ受信の要求を受け取ると, メッセージを受信側プロセスのバッファにコピーし, アドレスを返す.

(9) プロセスが通信を終え, ポートを解放する場合は, ポート番号を指定してポートの解放を要求する.

3.3 共有メモリによる IPC の効率化

Lavender では, コピーを伴う IPC を, 共有メモリを使用することで実行時に効率化する. これによって IPC のオーバヘッドを軽減することができる.

コピーによる IPC では, メッセージの送信時と受信時にコピーが 2 回発生するため, オーバヘッドが大きい. IPC ライブラリを使用した場合, メッセージは共有メモリに置かれ, IPC サーバの呼び出し回数と, メッセージのコピー回数が減少する. また, 共有メモリは共有しているプロセス間のみで参照することができるので, メッセージを保護することもできる. 以下に, IPC 機構の効率化の方法について述べる.

(1) IPC サーバは, 通信を開始しようとする双方のプロセスが同じノードに存在しているかを調べる.

(2) 同じノードに存在している場合, IPC サーバは, 通信しようとしている双方のプロセスのメモリ空間内に IPC 用の共有メモリを確保する. また, 共有メモリのアドレスを双方のプロセスに通知する.

(3) 送信側プロセスがメッセージを送信する場合, IPC ライブラリを介して共有メモリにメッセージを用意し, IPC サーバに送信の要求を出す.

(4) IPC サーバは受信側プロセスにメッセージの到着を通知する.

(5) 受信側プロセスは, IPC サーバを介さずに共有メモリからメッセージを読み込む.

4 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. このため, ハードウェアを直接制御する部分以外は, 他のシステムサーバプロセスと同じ構造を持つ.

4.1 デバイスドライバの内部構成

Lavender のデバイスドライバは, ライブラリとして提供されるいくつかのルーチン群と, ユーザが作成し, 登録するルーチン群から構成される. これらのルーチン群の動作を図1に示す.

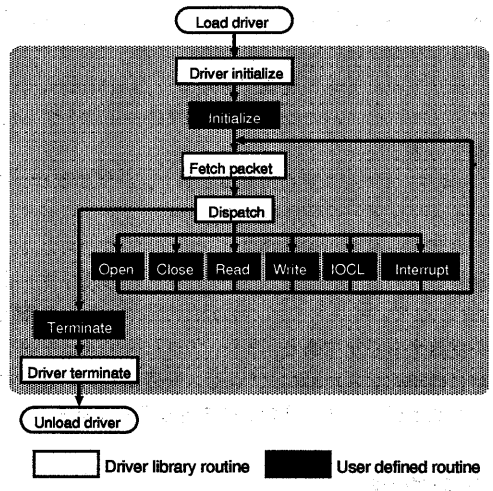

図 1 デバイスドライバの動作

デバイスドライバライブラリの構成
Lavender のデバイスドライバライブラリは, 以下のようなルーチンから構成される.

- **Driver initialize routine**
  ドライバのロード直後に実行される. 通信用ポートの取得や, デバイスドライバのネームサーバへの登録などを行う. このモジュールの実行後に, ユーザ定義の初期化モジュールが実行される.

- **Fetch routine**
  ポートから, ドライバへの処理要求が記述されたパケットを取り出し, 記述されている内容を解析する.

- **Dispatch routine**
  Fetch routine の解析結果に基づき, 各処理ルーチンに実行を移す.

- **Driver terminate routine**
  ユーザ定義の終端処理ルーチンの実行後に実行される. 使用していたポートを解放し, デバイスドライバのネームサーバからの削除などを行う.

ユーザ定義ルーチンの構成
Lavener のデバイスドライバは, 以下のようなルーチンから構成される.

- **Initialize Routine**
  デバイスドライバは生成されると, まず初期化手続きを実行する. ここで, 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, 変更許可をカーネルに依頼する. 物理メモリ空間のマップや, 外部のシステムサーバやカーネルとの通信用のポートを取得する. また, デバイスの初期化, ユーザが作成したルーチンの登録を行う. ドライバが割り込みを処理する必要がある場合は, 割り込み処理ルーチンをカーネルに登録する.

- **Terminate Routine**
  デバイスドライバの終了時に実行される. 各種リソースの解放などをこのルーチンで行う.

- **Open Routine**
  デバイスのオープン時に実行される.

- **Close Routine**
  デバイスのクローズ時に実行される.

- **Read Routine**
  デバイスからのデータ読み込み時に実行される.

- **Write Routine**
  デバイスへのデータ書き出し時に実行される.

- **I/O Control Routine**
  read, write 以外のデバイスの操作は, このルー

チンで行う．実際の処理内容はドライバに依存する．

- **Interrupt Routine**
  デバイスからの割り込み発生時に実行される．

4.2 割り込み制御機構

割り込みは，割り込み制御部，デバイスドライバ群などにより処理される．割り込みが発生した場合の動作を以下に述べる．

(1) システムコール，例外，デバイス割り込みなどの要因で割り込みが発生すると，カーネル内部の割り込み取得部に制御が移行する．このモジュールはニュークリアス層にあり，Lavender の動作しているハードウェアのアーキテクチャに依存している．

(2) 割り込み取得部には，カーネル内部のモジュールにある割り込み処理手続きが，それぞれの割り込み要因に対して登録されている．スケジューリングマネージャが使用するタイマ割り込みや，メモリマネージャが処理するページフォルトなどの割り込みは，それぞれのモジュール内の割り込み処理手続きが呼ばれ，そこで必要な処理がなされる．

また，デバイスドライバを必要とする周辺装置からの割り込みの場合は，対応するデバイスドライバの割り込み処理ルーチンに処理を移す．

ここで対応するデバイスドライバが，ユーザモードドライバの場合は，割り込みプロセス間通信変換部が，それをプロセス間通信のメッセージに変換する．このメッセージには，割り込みの内容が記述されている．その後プロセス間通信サーバに，このメッセージの送信を要求する．また，スケジューリングマネージャのスケジューリングキューに，この割り込み処理をするデバイスドライバのプロセスを押入する．

(3) プロセス間通信サーバがメッセージの到着をデバイスドライバのプロセスに通知して，割り込み処理から抜ける．

(4) スケジューリングマネージャがデバイスドライバのプロセスを実行状態にする．

(5) 実行状態になったデバイスドライバのプロセスは，割り込み要因が記述されたメッセージを受け取り，必要な処理を行う．

(6) デバイスドライバは必要な処理が終ると，必要があればシステムサーバやカーネル内のモジュールに結果を送信する．

割り込み取得部にある割り込み処理手続きのテーブルには，割り込み処理ルーチンが動的に登録，削除される．

5 おわりに

本論文では，Lavender の IPC 機構とデバイスドライバの構成について述べた．

IPC 機構については，共有メモリを利用した，プロセス間通信の実行時における効率化を中心に据えて設計，実装を進めている．

また，カーネル内部にあったデバイスドライバを，ユーザプロセスとしてカーネル外に出すという設計方針であるが，動作時のオーバヘッドを考慮に入れ，速度低下を最小限に抑えられるよう考慮した．また，ソフトリアルタイムが達成できるよう，LKM 形式でのデバイスドライバの動的な追加，削除を可能とする機構を用意した．

現在，プロセス間通信機構と割り込み制御機構，デバイスドライバの性能検証のために，コンソールドライバ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，割り込み制御部及びカーネルへの，デバイスドライバの動的な追加，削除の機構を実装する予定である．また，プロセス間通信を実行時に効率化するために，プロセス間通信サーバとプロセス間通信ライブラリの拡張を行う予定である．

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).