システムソフトウェアと
オペレーティング・システム 79－4
（1998．8．6）

マイクロカーネルLavender上のファイルシステムサーバの構築と応用

市岡 耕平† 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，階層化インタフェースを実装し，ポリシとメカニズムの分離という特徴を持つ．本論文では，Lavender上で動作するファイルシステムサーバの設計と，その応用について述べる．Lavender上のファイルシステムサーバは，ファイルマッピング方式でデータをクライアントに渡す．この方法により，ファイルシステム自体の柔軟性が高くなり，ページングや分散共有メモリなどへの応用が可能になる．

Design of File System Server on Lavender Micro Kernel and Its Applications

Kohei Ichioka† 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 layered interface and is featured separation of policy and mechanism. In this paper, the design of file system server on Lavender micro kernel and its applications are described. The file system server on Lavender micro kernel provides clients with data by the file-mapping mechanism. In this way, the file system server can be applied to not only data storage but also a paging system, distributed shared memory, and so on.

1 はじめに

従来のOSでは, その機能の大部分をカーネル内に実現しているため, カーネルのサイズが大きくなっている. また, カーネルがユーザにとって必要なサービスを実現していない場合や, 逆に不必要なサービスを含んでいる場合がある. さらに, サービスを追加, 削除, 変更することが困難である. 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, 拡張を容易に行うことができる. これにより, 従来のカーネルの問題点を解決することができる.

我々は, 以上の背景からマイクロカーネルLavenderを中心としたOSを構築している. Lavender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 [1] [2].

ファイルシステムにおいては, ファイルシステムサーバを追加することで, さまざまなファイルシステムフォーマットや, ネットワークファイルシステムのプロトコルに対応することができる. そのためには, ファイルシステムサーバのインタフェースを統一する必要がある. 本論文では, 特にファイルシステムサーバの設計とその応用について述べる.

従来のファイルシステムでは, ファイルシステム内のバッファとクライアントプロセスのデータ領域との間でデータを複写する方法でサービスを提供している. この方法では, クライアントとサーバの独立性が保たれるが, 柔軟性は低く, ファイルシステムの利用範囲が限定される. また, データを複写するため, 効率が悪い. ファイルマッピング方式を用いることで, 柔軟性を高め, ファイルシステムをページングや分散共有メモリなどに使用することが可能になる.

本論文では2章でLavenderの構成と特徴を述べ, 3章でファイルシステムサーバの設計, 4章でファイルシステムサーバの構成, 5章でファイルシステムサーバの応用について述べる.

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, カーネルで実現されなければならない最小限の機能を持つ. システムサーバは, ユーザアドレス空間に配置され, OSに必要な機能を実現する. Lavenderと複数のシステムサーバが協調し, OSとして機能する.

Lavenderは, 以下に示す特徴を持つマイクロカーネルである.

- ポリシとメカニズムの分離
- 階層化インタフェース
- ユーザカスタマイズ可能

(1) ポリシとメカニズムの分離

Lavenderは, システムサーバによるカスタマイズを可能にするために, ポリシとメカニズムの分離という特徴を持っている. 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である. Lavenderは, ポリシとメカニズムの分離を, 階層化インタフェースとシステムサーバにより実現している.

(2) 階層化インタフェース

Lavender内部の構成を図1に示す. Lavenderマイクロカーネルは, 内部にメモリ管理, プロセス管理, スケジューリング, 割り込み管理, プロセス間通信の機能を持つ. これらの機能は以下に示す3つの層に分割されている.

- **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, メカニズム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, システムサーバに対しメカニズムを提供する.
- **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, 従来のOSのシステムコールに相当する機能を提供する.

(3) ユーザカスタマイズ可能

従来のOSでは, カーネルの機能を変更する場合, カーネルのソースコードを変更し, カーネルを再構築する必要がある. Lavenderでは, カーネルとは独立したシステムサーバを用いることで, カーネルの機能を変更することを可能にしている.

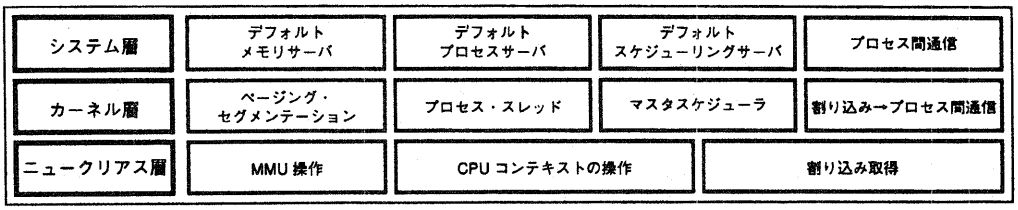

図 1　Lavender 内部の構成

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のポリシを実現する. システムサーバを置き換えることで, カーネルに変更を加えずに, OS としてのポリシを変更, 拡張することが可能である. システムサーバは, 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利用することができる.

3　ファイルシステムサーバの設計

Lavender マイクロカーネルには, ファイルシステムは含まれない. ファイルシステムは, ファイルシステムサーバと呼ばれるシステムサーバ群により実現される. ファイルシステムサーバを置き換えることにより, さまざまなファイルシステムフォーマットや, ネットワークファイルシステムのプロトコルに対応することができる. そのためには, ファイルシステムサーバのインタフェースを統一する必要がある. 本章では, ファイルマッピング方式を用いたインタフェースを設計することにより, ファイルシステムサーバの柔軟性を高めることについて検討する.

3.1　ファイルマッピング方式

ファイルマッピング方式とは, クライアントプロセスの論理アドレス空間にファイルをマップすることによって, ファイルに対するアクセスを可能にする方式である. ファイルがオープンされると, クライアントプロセスの論理アドレス空間に指定された大きさの領域が確保され, そのアドレス空間にファイルの内容の一部, または全部が見えるようになる. クライアントは, マップされた論理アドレスにデータが読み込まれているかのようにアクセスすることが可能である. 実際には, ファイルがオープンされたときにデータが主記憶に読み込まれているわけではなく, 必要に応じてファイルシステムサーバが読み込む.

ファイルマッピング方式は, ページングシステムでプロセスのデータ領域をページイン, ページアウトさせるのと同じ方法で実現される. ファイルマッピング方式はページングシステムをファイルのアクセスに応用したものといえる.

従来の OS では, このようにメモリ管理と深く関連した操作をメモリ管理モジュールの外から行うのは困難である. しかし, Lavender では階層化インタフェースを用いることにより容易に実現することができる.

3.2　ファイルキャッシュ

ファイルマッピング方式は, キャッシュの取り方に影響を与える. 従来のファイルシステムでは, ローカルディスクに対しては, ディスクのブロック番号で管理するバッファキャッシュを用いるのが一般的である [3]. バッファキャッシュ内のデータを直接クライアントプロセスが操作することはできない. バッファキャッシュ内のデータはどのファイルのものか分からないため, クライアントプロセスがそのデータにアクセスする権限があるか分からないからである. そのため, バッファキャッシュはファイルマッピング方式には適さない.

ファイルマッピング方式に適合したキャッシュとして, 以下の方法が考えられる. ファイルを主記憶のページサイズごとに分割し, ファイルの ID とファイル内のオフセットで管理するファイルキャッシュを用いる. ファイルキャッシュのデータは, ファイルごとに管理されているので, そのファイルにアクセスする権限を持つプロセスに対しては直接データを操作させることができる. キャッシュに使用している主記憶のページを直接クライアントのアドレス空間に割り当てることが可能なので, データの複写を必要としない.

3.3 メモリマッパ

本ファイルシステムサーバでは, キャッシュの柔軟性を高めるために, ファイルシステムサーバとは独立したメモリマッパを使用する. メモリマッパとファイルシステムサーバが協調してファイルマッピングを実現する. クライアントは, ファイルシステムサーバに要求を出すときに, メモリマッパを指定する. メモリマッパが提供する機能を以下に示す.

- クライアントの論理アドレス空間にページを割り当てる.
- キャッシュのポリシを提供する.
- クライアントのアクセスを監視する.
- クライアントが書き込んだページをファイルシステムサーバに伝える.

ファイルシステムサーバは, ファイルマッピングのために以下に示す機能を持つ.

- キャッシュに使用する物理ページを確保, 解放する.
- ディスクのデータを読み書きする.
- キャッシュのメカニズムを提供する.

メモリマッパは, 必要に応じて各ページの最終アクセス時間をファイルシステムサーバに通知する. これは, 実際の最終アクセス時間である必要はない. ファイルシステムサーバは, システム全体の物理メモリが不足すると, 最終アクセス時間の古いページを解放する. メモリマッパは, 通知する最終アクセス時間を調節することで, キャッシュのポリシを実現する.

また, リアルタイムアプリケーションのために, ファイルシステムサーバはページをロックする機能を持つ. リアルタイムアプリケーションを実行させるには, 処理時間が予測可能である必要がある. しかし, ディスク装置へのアクセスに必要な時間を予測するのは困難である. また, 予測できたとしても, 非常に長い時間を要する. そのため, アプリケーションの使用しているページをファイルシステムサーバが解放すると, リアルタイム性を保証できない. このような状況に対応するため, ファイルシステムサーバは, メモリマッパの要求を受けると, ページを解放しないようにする.

3.4 従来の方法との比較

ファイルマッピング方式とファイルキャッシュの利点を以下に示す.

- キャッシュとクライアントの間の複写が不要になる.
- 用途に応じたメモリマッパを使用することにより, アクセス特性に応じたキャッシュアルゴリズムを使用することができる.
- 柔軟性が高く, 単なるデータの保存以外にも使用できる.

また, ファイルマッピング方式とファイルキャッシュには, 以下に示す欠点がある.

- メモリ管理機構と密接な協調作業が必要になる.
- ディスクブロックで管理するバッファキャッシュに比べ, ファイル単位で管理するファイルキャッシュでは, 実現するための機構が若干複雑になる.

ファイルマッピング方式は, ページテーブルの操作など粒度の小さいメモリ管理を行わなければならない. 一般には, このようにメモリ管理機構の外部から粒度の小さいメモリ管理を行うのは困難である. しかし, 本ファイルシステムサーバは, Lavender の階層化インタフェースを用いて比較的容易に実現することができる.

従来のバッファキャッシュは, ディスク上のすべてのデータに対してキャッシュを行うことができる. ファイルキャッシュで, 同様にすべてのデータを扱うのは, そのための機構が必要である. この機構については, 4.2 節で述べる.

4 ファイルシステムサーバの構成

現在構築しているファイルシステムサーバと関連モジュールを, 図 2 に示す. 図 2 では, プロセス間通信を行うプロセス同士を線で結んである.

ファイルシステムサーバは, ディスクドライバ, ファイルサーバ, ディレクトリサーバの 3 つの層で構成される. ファイルサーバとディレクトリサーバはともにディスクの論理構造を解釈するが, 機能ごとに 2 つに分割できる [4].

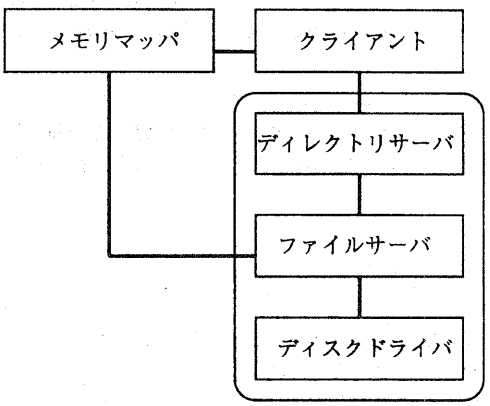

図 2　ファイルシステムサーバの構成と関連プロセス

4.1　ディスクドライバ

ディスクドライバは, I/O 権限を持つユーザプロセスとして実現される. ディスクドライバは, ディスク装置の記憶領域をブロック番号で識別し, ファイルやディレクトリの構成には関与しない.

4.2　ファイルサーバ

ファイルサーバは, ファイルキャッシュを管理するメカニズムを提供し, メモリマッパと協調してファイルマッピングを実現するサーバである. ファイルサーバは, ファイルを番号で識別し, ディレクトリの木構造には関与しない.

従来のバッファキャッシュは, ディスク上のすべてのデータに対してキャッシュを行うことができる. ファイルキャッシュでは, ファイルの内容に対してキャッシュを行うことになる. ディスクには, ファイルの内容だけでなく, ファイルの属性や, ディスク上の配置などの情報も書き込まれる. このような情報も, ファイルの内容と同様にキャッシュする必要がある.

このために, ファイルサーバは, ファイルの属性や配置などの書き込まれているディスク上の領域を 1 つのファイルとして扱う. このファイルを, ファイルサーバ自身のアドレス空間にマップする. この方法により, ファイルの属性や配置などの情報も, ファイルの内容と同様にキャッシュを行うことができる.

4.3　ディレクトリサーバ

ディレクトリサーバは, ファイルを名前で扱うための機能を提供する. ファイルサーバはファイルを数字で管理しているので, ファイルを名前で使用するためにディレクトリサーバが必要である.

ディレクトリサーバは, ファイルの番号を調べるためにディレクトリファイルを使用する. ディレクトリファイルは, ファイル名とファイルの番号の組を保持している. ディレクトリサーバは, ディレクトリファイルから目的のファイルの番号を検索する. ファイルサーバは, ディレクトリファイルと一般ファイルを同じように扱う. ディレクトリサーバは, ファイルマッピング方式でディレクトリファイルにアクセスする.

ディレクトリファイルが不正に書き換えられると, ディレクトリ構造に矛盾が生じる可能性がある. これを防ぐために, クライアントの要求はすべてディレクトリサーバを通して行う.

4.4　ライブラリ

従来の read/write によるアクセスを行うソースプログラムも使用できるように, read/write の動作を行うライブラリを用意する予定である. また, ストリーム入出力関係のライブラリをファイルマッピング方式用に書き換えることで, 既存の多くのプログラムを効率的に実行することができる. ストリーム入出力関係のライブラリでは, クライアントプロセスのデータ領域にバッファを用意し, このバッファを経由してファイルにアクセスする. ファイルのマップされたアドレスを直接利用することで, クライアントプロセスのデータ領域にバッファは不要になる.

5　ファイルシステムサーバの応用

ファイルマッピング方式によるファイルシステムサーバは, 柔軟性が高く, さまざまな応用が考えられる.

5.1　ページング

ファイルマッピング方式は, ページングによる仮想記憶をファイルアクセスに応用したものである. そのため, ファイルマッピング方式によるファイルシステムを用いて, ページングによる仮想記憶を実現するのは容易である. ファイルのマップされたアドレス空間を, プロセスのコード領域やデータ領域として使用す

ることで, ページングによる仮想記憶を実現することができる.

多くのOSでは, 同じプログラムファイルをもとに複数のプロセスが生成されたときは, それらのプロセス間でコード領域を共有する. これにより, メモリを効率的に使用することができる.

また, 長時間使用されていないコード領域のページをディスクのスワップ領域にページアウトしないオペレーティングシステムもある. 長時間使用されないコード領域のページは, 内容を破棄し, 必要なときに再びファイルから読み出す. この機構により, ページアウトの回数が減少する.

ファイルマッピング方式によるページングでは, これらの機能を簡単に実現することができる. プログラムファイルのファイルマッピングを作成し, それをそのままコード領域にするだけで, これらの機能が実現される.

5.2 分散共有メモリ

ファイルマッピング方式によるネットワークファイルシステムにより, 分散共有メモリを実現することができる. 複数のノードで共有しているファイルのファイルマッピングは, 複数のノードで共有するメモリとして使用することができる.

5.3 パラメタマーシャリングの効率化

サーバプロセスに対する要求は, プロセス間通信を用いて行われる. 要求にはさまざまな種類のパラメタが用いられるため, これらのパラメタをバイト列に変換し, プロセス間通信を利用できるようにする必要がある. この処理をパラメタマーシャリングという.

パラメタとしてポインタを使用する場合, ポインタそのものではなく, ポインタの指すデータを渡す必要がある. これには, ポインタの指すデータをプロセス間通信のメッセージに詰めるのが最も簡単である. しかし, データが巨大な場合は, メッセージが大きくなり, プロセス間通信のオーバーヘッドが大きくなる. 特に, 他のノードのサーバを呼び出すときには, ネットワークトラフィックにも影響を与える.

このとき, パラメタとして指定されたポインタがネットワークファイルシステムのファイルがマップされたアドレスを指していて, 送信先のノードが同じファイルのデータを持っている場合が考えられる. このような場合には, ポインタの指すデータそのものを送信する必要はない. ポインタから, ファイルの識別子とファイルの先頭からのオフセットを調べ, それらを送ればよい.

現在, Lavender 上に, パラメタマーシャリングを行う汎用モジュールの構築について検討している. その際には, 上記の効率化が可能なように設計する予定である.

6 おわりに

本論文では, Lavender 上に構築するファイルシステムサーバについて述べた.

ファイルマッピング方式により, ファイルシステムサーバの柔軟性が高まり, さまざまな応用が可能になる. また, ファイルマッピング方式に適合したファイルキャッシュを用いることで, データの複写を省くことができる. ファイルシステムサーバから独立したメモリマッパを用いることで, クライアントのアクセス特性に応じたキャッシュが可能になる.

ファイルマッピング方式は, ページングによる仮想記憶をファイルアクセスに応用したものである. そのため, ページングによる仮想記憶や, 分散共有メモリなどに利用することができる.

現在, カーネル内にディスクドライバが実装されている. 継続して, ファイルサーバやディレクトリサーバを実装する予定である. また, メモリマッパやライブラリの整備も行う予定である.

参考文献

[1] 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).

[2] 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).

[3] Maurice J.Bach 著 坂本 文, 多田 好克, 村井 純 訳: "*UNIX* カーネルの設計", 共立出版 (1991).

[4] A·S·タネンバウム 著 水野忠則, 鈴木健二, 宮西洋太郎, 佐藤文明 訳: "分散オペレーティングシステム", プレンティスホール (1996).