システムソフトウェアと
オペレーティング・システム 76－11
（1997. 8. 21）

カーネルの発展性と安全性に関する一考察

多田 好克, 中村 嘉志, 林 隆宏.
電気通信大学 大学院情報システム学研究科
〒182 東京都調布市調布ヶ丘 1-5-1
E-mail: {yoshi,nmura,lannet}@spa.is.uec.ac.jp

あらまし　近年、カーネルの発展性 (extensibility) に関する要求が高まっている。これは、新しい概念のハードウェアの出現や様々な応用プログラムからの要求に対応するためであると考えられる。この要求に応える方法としてはマイクロカーネルとカーネルモジュール取込み (lkm) が考えられるが、前者は発展部実行時の処理効率に、後者は発展部開発時の安全性に、それぞれ問題がある。本稿では、これらの欠点を軽減する保護付きモジュール取込み (plkm) を提案し、その実現法を議論する。

和文キーワード　カーネル、発展性、発展機構、モジュール化、lkm、保護、実行効率

Kernel's extensibility and safety reconsidered

Yoshikatsu Tada, Yoshiyuki Nakamura, and Takahiro Hayashi.
Graduate School of Information Systems
The University of Electro-Communications
1-5-1 Choofugaoka, Choofu-shi Tokyo　182　Japan
E-mail: {yoshi,nmura,lannet}@spa.is.uec.ac.jp

Abstract　Kernel program extensibility is required because of the emergence of hardwares with new concepts, and a need of application softwares. Microkernels and loadable kernel modules have been developed recently, but the former is suffering the unefficiency of the execution phase while the latter is suffering the difficulty of the debugging phase. In this paper, we propose a protected loadable kernel module (plkm), which keeps the efficiency and the safety of the existing kernel and the extended loadable modules.

英文 key words　kernel extension, loadable kernel module, protection, efficiency

1 はじめに

ＯＳの核となるカーネルは、進化・発展させにくいソフトウェアの１つである[1]。カーネルの発展には、ＯＳ上で作動する応用プログラムとのインタフェースを維持し、利用者が抱く仮想計算機のイメージを変化させてはならないという制約が付いて廻る。四半世紀も前に設計されたUnixが現在でも広く使われている理由の１つは、この制約によると考えられる。

また、カーネルの発展には特有の難しさが存在する。カーネルの改良、虫取り中には、カーネルの提供する保護機能や入出力機構が使えない。そのため、虫の再現性が低く、また、虫に関する十分な情報を得ることも難しい。

他方、カーネルは発展しなければならないソフトウェアでもある。ハードウェアの日進月歩の進化と、利用者の要求の多様化に追従しなければならないからである。ビットマップディスプレイやマウス、ネットワークインタフェースなど、Unix開発後に出現したハードウェアは数多い。また、応用プログラムに即したファイルシステムやメモリ管理方針が必要とされているし、場合によっては異なるＯＳを１台の計算機上に混在させたいこともある。さらに、並列・分散処理、実時間処理、モーバイル／ユビキタス環境などの新たな応用分野も出現し、カーネルはこれらの要求にも応えなければならない。

本稿では、以後、カーネルの機能を拡張するカーネル発展の可能性について議論する。まず、カーネルの発展性を、

- 発展部開発時の安全性；
- 発展部実行時の実行効率；

の２つの観点より評価する。そして、我々が提案する「保護付きモジュール取込み」(plkm: protected loadable kernel module)の概要を紹介する。

[1]この研究の一部は、文部省科研費重点領域「ソフトウェア発展(09245211)」の援助を受けて行われている。

2 本研究の目標

現在、発展機構を備えたカーネルとしてはマイクロカーネル方式が主流である。しかし、マイクロカーネルを利用した実現法では実行時のオーバヘッドが無視できず、作成したシステム上の応用プログラムで1.5倍程度の処理時間が必要になると言われている（次章の議論を参照）。

一方、近年活用され出したカーネルモジュール取込み(lkm: loadable kernel module)によるカーネル発展機構では、カーネル拡張時の保護機構が不十分であり、マイクロカーネルで提供される各種サーバのような複雑な機構を付加する場合には開発時の生産性が低下する。

そこで本研究では、まず、既存のカーネル発展機構を調べその問題点を明確にする。次に、lkmを改良し、保護機構を兼ね備えたカーネル発展機構plkmを提案する。plkmは、まだ実体を伴わない概念のみの機構である。本稿の後半では、plkm実現に関するアイデアも紹介する。

3 マイクロカーネル v.s. カーネルモジュール取込み

3.1 マイクロカーネルの欠点

マイクロカーネル上にたとえばUnixサーバを作成した場合、システムコール時のオーバヘッドが無視できない[1]。80486 DX50上に実現された第１世代のマイクロカーネルでは、Unixサーバが実行するシステムコールのオーバヘッドは115 μ秒である。

一方、同一ハードウェア上に直接実現したUnixでのシステムコールのオーバヘッドは20 μ秒程度なので、この違いは無視できない。プログラムの性質によるが、実用的な応用プログラムによる測定でも実行時間で1.5倍程度の処理時間が必要だと言われている（表1参照）。

表1: 応用プログラムの実行時間比

応用プログラム	実行時間比（Unix：1）
sed	1.66
gcc	1.45
compress	1.34

このオーバヘッドの原因の1つは、応用プログラムとサーバ間のIPCである。Mach等のいわゆる第1世代のマイクロカーネルではIPCに100 μ秒程度を要する。これは、たとえばExokernel[2]等の第2世代のIPCでは10 μ秒程度になったと言われているが、単層カーネル(monolithic kernel)のシステムコールのオーバヘッドである関数呼出しに比べると、まだ1桁の違いがある。

3.2 カーネルモジュール取込みの欠点

lkmでは、取り込まれたモジュールは、既存の単層カーネルのモジュールと（ほぼ）同じオーバヘッドで作動する。しかし、マイクロカーネルのサーバ作成に比べて虫取りが難しく、作成中のモジュールがＯＳのハングアップを招くことも少なくない[2]。

Spin[3]ではコンパイラと協調してカーネルの既存部分の保護を図っているが、その方法論には限界があり、どこまで有効か疑問が残る。また、Javaのアプローチの延長として、取り込んだモジュールはインタプリティブに実行し保護を図るという方法も考えられるが、実行時のオーバヘッドは避けられない。実行時にはコンパイルして処理効率を上げるというアイデアもあるが、コンパイルによって実行環境が変わるため新たな虫発生の可能性が高い。

4 保護付きモジュール取込み

我々が提案するplkmの基本的なアイデアは、開発時に取込みモジュールを走らせる場合は保護を強化し、虫取りが一段落した時点でその保護を取り除き高速実行させようというものである。

plkmを使って作成したモジュールは、僅かな意味の変更のみで保護なしの取込みモジュールに変更できる。インタプリタによる保護によって作成した取込みモジュールは、コンパイルによる意味変更の危険を伴うが、plkmではそれが少ない。plkmを使うことによってlkmの問題点が完全に除去できるわけではないが、開発時の効率が向上し、実行時の処理効率も確保できる。

plkmの実現においては、発展部開発時に、たとえば、以下のような制限を設けることが考えられる。

- カーネルメモリへの書込みは、チェックを受ける；
- 入出力装置等への読み書きは禁止し、実行時に既存のカーネルが処理を代行する；
- 既存のカーネルコードへの制御の移行時には妥当性をチェックする；
- 既存のカーネルコードへの制御の移行時には引数をチェックする[3]；

[2] FreeBSDの`modload`コマンドでは、取り込んだモジュールに制御を移す前に`sync`を実行している。ご愛嬌ではあるが、発展部の開発者としては笑えない。

[3] 実現するためにはカーネル内に引数の型、範囲などの情報を置く必要がある。しかし、これはカー

表 2: 発展部とのインタフェース

機　　能	対 マイクロカーネル（と既存サーバ群）
制御の譲渡	システムコール、プリエンプション（IPC）
制御の獲得	アップコール、システムコール（IPC）
データの付与	システムコールの引数（メッセージ）
データの参照	システムコールの戻り値、アップコールの引数（メッセージ）
機　　能	**対 lkmの既存カーネル部**
制御の譲渡	関数呼出し
制御の獲得	カーネルフック
データの付与	関数呼出しの引数、大域変数への書込み
データの参照	大域変数の参照

また、一般に既存のカーネルコードは発展部のデータをアクセスしないので、

- メモリ保護機能を活用できる；

という期待もある。

5　plkm の機能的な拡張

ここまでは、マイクロカーネルと lkm とに関する、

- 発展部開発時の虫取り効率；
- 発展部実行時の処理効率；

を中心に plkm の実現法を議論してきた。以下では、はもう 1 つの観点として plkm の機能的な拡張の可能性について議論する。つまり、plkm では「発展部にどのような機能を提供できるか？」を考える。

マイクロカーネル（と既存サーバ群）または既存カーネル部分と、サーバまたは取込みモジュール（つまり発展部）とのインタフェースは、表 2のようになる。

マイクロカーネルでは、あらかじめシステムコールとして用意された機能しか利用できない。マイクロカーネルの発展部として実現できる機能は、この制限を受けることになる。

また、カーネルモジュール取込みでも、既存のカーネルにあらかじめフックが用意されていなければ、発展部に制御を移すことができない。たとえば、FreeBSD の lkm では、

- システムコール実行時；
- vnode を参照した時；
- デバイスドライバのインタフェースを継承；
- その他（たとえば、スクリーンセーバ用のタイムアウトフックなど）；

といったフックが用意されている。plkm ではメモリ保護機構やコードの置換えによって、より柔軟なフック機能の提供を試みる[4]。

ネルの肥大化を招き好ましくはない。我々はチェック用の取込みモジュールを用意し、この問題を回避できると考えている。

[4] これは、カーネル内にデバッガを設置するようなものと考えられる。plkm を使えば、このような機能も動的に取り外し可能なモジュールとして実現できる。

さらに、lkmではサーバもしくはデーモンのように、定期的に作動するようなモジュールの追加が難しい。つまり、既存カーネル部分とのスケジューリングに関する方法論が確立されていないのである。plkmでは定期的に制御を割り当てられるプリエンプティブなスレッドを用意する[5]。

また、lkmに対する不満の1つとして、カーネルの構造を知らなければ取込みモジュールを作成できない、という意見がある。カーネルを拡張するプログラマはカーネルの動作ぐらい知っているべきだ、という考えもあるが、カーネルの発展性を考えるならば、デバイスドライバの追加のように必要なインタフェースの資料と背景となる少しの知識だけで発展部をプログラミングできることが望ましい。このためには、カーネルのモジュール化のような作業も必要になると考えている。

6 おわりに

本稿では、カーネルモジュール取込みを発展させ、

- 発展部開発時の虫取り効率の向上；
- 発展部実行時の処理効率の向上；

を目指した保護付きモジュール取込み機構(plkm)の特長を示し、その実現可能性を議論した。基本的なアイデアは、開発時には実行速度の低下を覚悟で各種の保護を行ない、実行時にはその保護を外して高速実行するというものである。ただし、2つの動作状態において、意味の違いを小さくできるかどうかが研究の鍵になると考えている。

今後は、

- 具体的な実現法；

を確定し、

- 実際の処理系；

を作成した上で、

- 保護の有無による動作タイミングの違い；

などの意味の変化に対する堅牢さを評価し、

- より良いカーネル発展機構の提供；

を目指す予定である。

[5]カーネルから一定間隔で一定時間、制御を得られるようなフックを用意し、発展部用のスケジューラをplkmとして実現可能とする。

参考文献

[1] Liedtke, Jochen: Toward Real Microkernels, CACM, 39,9, pp.70–77 (1996). (谷口秀夫訳：真のマイクロカーネルへ向けて、bit、29,8、pp.11–21 (1997).)

[2] Engler, D., M.F. Kaashoek, and J. O'Toole: Exokernel, an operating system architecture for application–level resource management, Proc. of SOSP, pp.251-266 (1995).

[3] Bershad, B.N., S. Savage, P. Pardyak, et al.: Extensibility, safety, and performance in the Spin operating system, Proc. of SOSP, pp.267–284 (1995).