システムソフトウェアと
オペレーティング・システム 76－7
（1997．8．21）

新規ファイルシステムの開発におけるOSの多段階保護機構の必要性

光来 健一　千葉 滋　益田 隆司

東京大学大学院 理学系研究科 情報科学専攻
〒113 東京都 文京区 本郷 7-3-1

要旨

OSを拡張しやすくするためには、拡張によってOSが不安定にならないようにする機構が必要である。従来研究されてきた保護機構ではOSの安定性を保つことはできても、実行効率が悪い。そこで我々は段階的に保護の強さを変えることができる多段階保護機構を提案し、実際にファイルシステムに対して多段階保護機構を実現するシステムをNetBSD上に実装した。このシステムを使えば、ファイルシステムが不安定な時は保護を強くしてOSを守り、安定してくれば保護を弱めて実行効率を改善することができる。さらに我々はこのシステム上で実際にファイルシステムを作成し、実験によって多段階保護機構の有用性を示した。

Necessity of multi-level protection for developing new file systems

Kenichi Kourai, Shigeru Chiba and Takashi Masuda

Department of Information Science, Graduate School of Science,
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113

Abstract

Operating systems that are easy to extend should have a protection mechanism for preventing them from being unstable when they are extended. Previous protection mechanisms are workable for this aim, but not efficient. We propose multi-level protection, which allows users to change the strength of protection, and we have partially implemented it on the NetBSD operating system. The protection mechanism we have implemented makes it easy to develop a new file system; for example, the users can select stronger protection if the new file system is unstable, but they can weaken the strength of the protection and improve the execution performance as it gets stable. Also, we have actually developed two file systems with the proposed mechanism to show its usefulness.

1 はじめに

OSの柔軟性を高めるために拡張可能なOSがさかんに研究されている。しかし拡張したことによってOSが不安定になるようでは、決して拡張しやすいとはいえない。それゆえ拡張のために追加したモジュールの不正な振舞いのせいで、OS全体が不安定にならないようにする機構が必要である。

そのような機構として、我々は多段階保護機構を提案する。従来研究されてきた保護機構ではOSの安定性を保つことはできても、実行効率が悪い。そこで多段階保護機構は、モジュールの安定度に合わせて、段階的に保護の強さを変える機能を提供する。この保護機構では保護を弱めれば実行効率はよくなり、保護を強めれば実行効率はその分悪くなる。OSを拡張するためのモジュールは一般に信頼できるので、可能な限り保護を弱めて実行効率を改善した方がよい。どのくらい弱められるかはモジュールの安定度によるので、保護の強さは多段階用意されている。また保護の強さの変化による実装の差異は多段階保護機構が隠蔽し、ユーザがモジュールのソースコードを書き変える必要はない。

多段階保護機構の実現可能性を確かめるために、我々はファイルシステムに対して多段階保護機構を実現するシステムをNetBSD上に実装した。このシステムを使えば、ファイルシステムが不安定な時は保護を強くしてOSを守り、安定してくれば保護を弱めて実行効率をよくすることができる。このシステムはハードウェアによる保護だけでなく、ソフトウェアによる保護も利用することによって、全部で5段階の強さの保護を提供する。我々はこのシステム上で実際にファイルシステムを作成し、実験によって多段階保護機構の有用性を示した。

2 多段階保護機構

拡張しやすいOSを作る上では、拡張によってOSが不安定にならないように保護機構を工夫することが重要である。本節ではユーザがOSの安定性と実行効率を段階的に調節できるような保護機構の必要性を述べる。

2.1 拡張しやすいOS

Exokernel [1]やSPIN [2]のような拡張可能なOSがさかんに研究されている。拡張可能なOSでは、ファイルシステムやデバイスドライバのようなモジュールをOSに追加することによって、柔軟性を増し、性能をよくすることができる。しかし拡張可能なOSが必ずしも拡張しやすいOSというわけではない。例えば追加したモジュールが不正な振舞いをした時に、OSが異常に遅くなったり、しばらくして止まったりするなど、不安定になるようでは拡張しやすいとはいえない。しかしながら、拡張したことによってOSが不安定にならないようにする機構はまだ研究途上である。

完全な保護機構を使ってOSの安定性を保つのは比較的容易である。ファイルシステムなどのモジュールをカーネルに直接組み込まずに、UNIXのユーザプロセスとして実装すればよいからである。モジュールからカーネルのリソースへのアクセスはシステムコールを使って安全に行なわれる。また、モジュールは別のアドレス空間をもつ他のユーザプロセスのリソースにはアクセスできない。このように、UNIXの保護機構を使ってモジュールをカーネルや他のユーザプロセスから完全に切り離してしまえば、OSの安定性を容易に保つことができる。

しかしながら、ユーザプロセスを使う方法では保護が強力すぎるので、実行効率が非常に悪くなる。ユーザプロセスは信頼できないアプリケーションからOSを守るための保護機構だからである。信頼できないアプリケーションとは、アクセス権をもっていないリソースに故意にアクセスする、悪意をもっているアプリケーションのことを指す。UNIXの保護機構ではこのような信頼できないアプリケーションからOSを守るために、カーネルのリソースへのアクセスにはシステムコールを使うが、これが大きなオーバヘッドになる。システムコールを実行するとユーザ空間からカーネル空間へのコンテクストスイッチが起こり、終了するとカーネル空間からユーザ空間へのコンテクストスイッチが起こる。さらに、カーネル空間のデータをユーザプロセスが直接読むことはできないので、必要ならばカーネルはそのデータをユーザ空間にコピーしなければならない。

2.2 多段階保護機構の必要性

OSを拡張するためのモジュールにはUNIXのユーザプロセスが提供するような完全な保護は必要ではない。OSを拡張するためのモジュールは不正な振舞いをする可能性はあるが、悪意をもって作られているわけではないので、比較的信頼することができるからである。保護機構は、プログラムの誤りによるモジュールの不正な振舞いがOS全体を不安定にする事態を防げさえすればよい。

我々はOSを拡張するためのモジュールについては、保護の完全性を緩めて実行効率をよくするべきであると考える。このように完全性を緩めた保護を我々は弱い保護と呼んでいる。この弱い保護を使え

ば、完全な保護に必然的に伴なうオーバヘッドを減らして、実行効率を改善できる。例えばモジュールとカーネルとの通信を、保護のためにシステムコールを使って実装するのではなく、手続き呼び出しを使って効率よく実装できるかもしれない。しかしながら、モジュールごとに安定度は異なるので、全ての場合に最適な強さの保護というのは存在せず、弱い保護を一種類だけ提供しても十分ではない。

そこで我々は、モジュールの安定度に合わせて、段階的に保護の強さを変えられる多段階保護機構を提案する。この保護機構を使うとモジュールが安定していて信頼できる場合には、実行効率を考えて保護を非常に弱めることができる。逆にモジュールが不安定な場合は、実行効率よりも保護を優先させることができる。また保護の強さを変える時も、ユーザがモジュールのソースコードを変更する必要はない。保護の強さの変化によって保護の実装方法が変わっても、多段階保護機構がその違いを隠蔽するからである。

3 ファイルシステムのための多段階保護機構

前節で述べたアイデアの実現可能性と有用性を確かめるために、ファイルシステムに対して多段階保護機構を実現するシステムをNetBSD上に実装した。

3.1 応用例

多段階保護機構を使うと、ファイルシステムの開発の進み具合に応じて、保護の強さを変えることができる。開発の初期段階と完成後とで同じ強さの保護を使うのは無駄である。初期段階ではファイルシステムは不安定なので強力な保護が必要だが、完成後には不正な振舞いはほとんどないので保護は必要ないと考えられる。そこで、開発者はOSの安定性と実行効率のトレードオフを考えて、開発が進んでファイルシステムが安定するにつれて保護を弱める。そして最終的には保護をなくして、カーネルに最初から作り込んだファイルシステムと同じ程度に実行効率をよくすることができる。保護の強さの変更は、モジュールの再コンパイルまたは再リンクによって行なう。その際に開発者はソースコードを書き変える必要はないので、この実装方法によって開発がより煩雑になることはない。

また、ベンダが提供する、不安定かもしれないファイルシステムやその機能拡張などのモジュールを使う時にも、多段階保護機構が役に立つ。そのようなモジュールを使う時には、OS全体が不安定にならないように、最初はそのモジュールに強めの保護をかけて使うとよい。もし不安定で不正な振舞いをするようなら、そのモジュールはOS全体を不安定にすることなく安全に切り離すことができる。しばらく使ってみて安定しているようなら、より弱い保護に変えて実行効率がよくなるようにすればよい。

3.2 多段階保護機構の実装方法

我々が実装したファイルシステムのための多段階保護機構は、5段階の強さの保護を提供する。以下ではそれぞれの段階で、どのように弱い保護が実現されているかを述べる。

3.2.1 第0段階

まず始めに、第0段階として、OSの安定性を最大にするために、最も強力な保護を実現する方法を述べる。この方法は実行効率が非常に悪いので、実際には我々のシステムでは提供していない。しかし、残りの段階の実現方法はこの段階を基礎としているので、説明を簡潔にするためにここで述べる。

第0段階では、ファイルシステムはOSの安定性を保つために、ユーザプロセスとして作られ、カーネルから完全に切り離される。この利点はユーザプロセスとして作られたファイルシステムがどんなに不正な振舞いをしても、カーネルや他のユーザプロセスは全く影響を受けないという点である。

ユーザプロセスとして作られたファイルシステムはカーネルと通信するために、システムコールとアップコールを使用する。システムコールはカーネルのデータを得たり、カーネルの関数を実行したりするために使われ、アップコールはカーネルがユーザプロセスの関数を呼び出す時に使われる。NetBSDではアプリケーションがファイルに関するシステムコールを発行すると、まずVFS [3]という仮想的なファイルシステムが呼び出され、そこから各ファイルシステムに振り分けられる。カーネルからユーザプロセスの関数を直接呼び出すことはできないので、シグナルを使ってアップコールすることによって呼び出す。

3.2.2 第1段階〜第4段階

これから説明する第1段階から第5段階までが、我々のシステムが実際に提供している保護である。このうち第1段階から第4段階では、第0段階の実行効率を改善するために、ファイルシステムとカーネルの通信に共有メモリを使う。第0段階で実行効率が悪い原因は、ファイルシステムがカーネルの

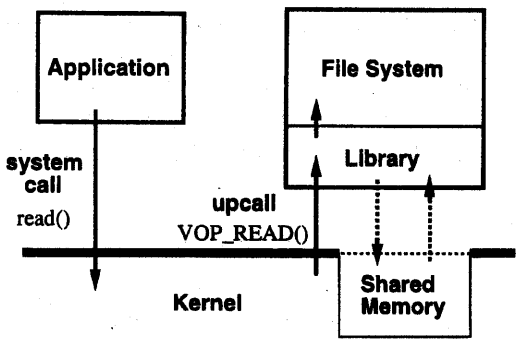

図 1: 第 1 段階から第 4 段階までのファイルシステム呼び出しの仕組

データを必要とする時に、毎回システムコールを使わなければならず、コンテクストスイッチやコピーのオーバヘッドが大きくなることである。そこでファイルシステムとカーネルの間に共有メモリを張り、ファイルシステムが必要とするカーネルのデータを共有メモリ上に置く。

しかし、ファイルシステムが直接共有メモリにアクセスすると危険なので、共有メモリへのアクセスは、必ず本システムが提供するライブラリを通して行われる。このライブラリはファイルシステムにリンクされ、共有メモリへのアクセスの隠蔽と保護を行なう。本システムでは第 1 段階から第 4 段階それぞれについて、異なる強さの保護を実装したライブラリを提供している。保護の実装方法の違いはライブラリが吸収するので、保護の強さを変える時に、プログラマがファイルシステムのソースコードを変更する必要はない。第 1 段階から第 4 段階でのファイルシステムの呼び出しの仕組を図 1に示す。

第 1 段階

第 1 段階で提供される保護は、ライブラリがアクセスする時以外は共有メモリをアンマップし、ファイルシステムはそのコピーにアクセスするというものである。ファイルシステムが、コピーされたカーネルのデータにアクセスする時には、共有メモリはアンマップされているので、不正なアクセスが OS を不安定にすることはない。ライブラリが隠している共有メモリをマップするためのキーを手に入れて、共有メモリをマップすれば不正なアクセスが可能だが、プログラムの誤りによってこのようなことが起こる可能性はほとんどない。第 1 段階では共有メモリのアンマップやコピーにかなりのコストがかかるので、第 0 段階と比べて実行効率はあまりよくならない。しかしライブラリがカーネルのデータにアクセスする際にシステムコールが必要なくなるので、ポインタや配列が扱いやすくなるという利点がある。

第 2 段階

第 2 段階で提供する保護は第 1 段階より弱められている。第 1 段階との違いは、共有メモリをアンマップする代わりに読み出し専用にしていることである。共有メモリをアンマップするのに比べて、読み出し専用にする方が高速に実行できるので、第 1 段階に比べて実行効率がよくなる。また、参照するだけで変更しないデータならば、コピーせずに共有メモリに直接アクセスすることができ、さらに実行効率がよくなる。しかし、不正に共有メモリを読むことができてしまうので、OS の安定性は多少失なわれるかもしれない。

第 3 段階

第 3 段階は第 2 段階よりもさらに弱い保護を提供する。第 2 段階と違って、共有メモリは常に読み書き可能なモードでマップされている。それゆえファイルシステムが共有メモリに誤って書き込めてしまい危険だが、依然としてファイルシステムはライブラリを通して共有メモリのデータのコピーにアクセスするので、ある程度は安全である。第 3 段階では、第 2 段階で行なっている共有メモリのアクセスモードの切り換えも行なわないので、非常に実行効率がよい。

第 4 段階

第 4 段階では共有メモリを全く保護しない。ファイルシステムは、第 3 段階のように共有メモリのデータのコピーにアクセスするのではなく、直接共有メモリにアクセスする。ただし、そのアクセスはライブラリを通して行ない、コンパイラがそれを共有メモリに直接アクセスするコードに置き換える。コピーをしないので、実行効率はさらによくなるが、ファイルシステムが共有メモリに直接アクセスすることによって、カーネルのデータを誤って破壊する可能性が高くなる。しかしながら、ユーザ空間とカーネル空間の間の保護は残っているので、共有メモリ以外の部分では安全性が保たれる。

3.2.3 第5段階

第1段階から第4段階までは、ファイルシステムはユーザプロセスとして作られていたが、第5段階では実行効率を最優先して、カーネルの中に組み込まれる。これによって、VFSはファイルシステムの関数をアップコールを使わずに、直接呼び出すことができるようになる。また、ファイルシステムはシステムコールを使わずに、カーネルの関数を直接呼び出せるようになるし、第4段階ではカーネルから共有メモリに一旦コピーする必要があったデータでさえも、コピーする必要がなくなる。

カーネルに組み込むことによって完全に保護がなくなってしまうが、カーネルに作り込みのファイルシステムとほぼ同程度の実行効率を得ることができる。ただし、保護の強さを変えてもソースコードを変更しないで済むように、保護の実装方法の差異を吸収する関数を使う必要があるので、そのオーバヘッドの分だけ実行効率が悪くなる。しかし、マクロ展開やインライン展開といったコンパイラ技術を使うことでこのオーバヘッドは軽減できると考える。

4 実験

多段階保護機構の実行時の性能を調べるために、2種類のファイルシステム、smfs (simple memory file system) と snfs (simple network file system) を我々のシステム上で作成し、実験を行なった。smfsはメモリファイルシステムであり、メモリ上にファイルシステムを作成して読み書きを行なっている。一方、snfsはNFS [4] のようなネットワークファイルシステムであり、クライアントは RPC [5] を用いてサーバと通信し、サーバがローカルディスクに読み書きをする。ただしローカルディスクへの書き込みは非同期に行なわれる。

この実験を行なった計算機はSPARCstation 5 (MicroSPARC2/85MHz, メモリ 32M) で、OSは多段階保護機構を組み込んだNetBSD 1.2である。またsnfsのサーバが使用するローカルディスクにはSunのSCSI 1.3G (5,400rpm) を使用した。

実験ではブロックサイズを512, 1K, 2K, 4K, 8K, 16K, 32K, 64Kbyteと変えて全体で64kbyteのファイルの転送をsmfs、snfsについて行ない、経過時間を保護の強さを変えて測定した。また、比較のために多段階保護機構を使わずにカーネルに直接smfs、snfsを実装し、同様の測定を行なった。

smfsにおける結果を図2に、snfsにおける結果を図3に示す。カーネルに直接作り込んだsmfsとsnfsはこの図ではほとんど重なってしまうので、図には

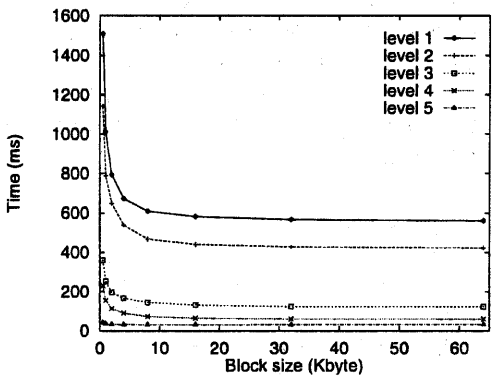

図 2: smfsにおける保護の強さ別ファイル転送時間

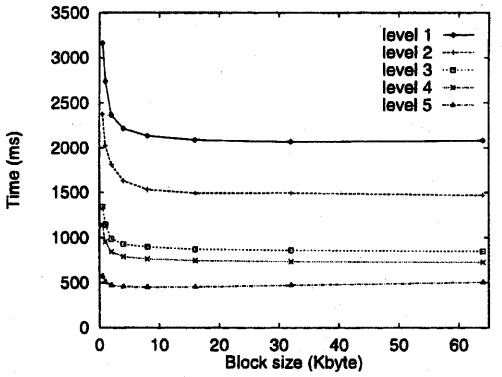

図 3: snfsにおける保護の強さ別ファイル転送時間

示していない。これら2つの図から、保護を弱めていくにつれて、実行効率が改善されていることがわかる。カーネルに直接作り込んだものに比べて、多段階保護機構のオーバヘッドはそれほど大きくない。第1段階の強い保護でもsmfsで15～16倍程度、snfsでは5～6倍程度で済んでいる。

第1段階から第4段階までの性能にはブロックサイズが大きく影響している。ブロックサイズが小さい時は、多段階保護機構のオーバヘッドが非常に大きくなっているが、ブロックサイズがある程度大きくなると、オーバヘッドはほぼ一定になっている。これはブロックサイズが小さい時には、アップコールやシステムコールが頻繁に起こるためだと考えられる。またsnfsのようにファイルシステム自体の処理量が多い場合には、相対的に多段階保護機構のオーバヘッドは小さくなる。

我々のシステムで最も保護の弱い第5段階のファ

イルシステムと、カーネルに直接作り込んだものとを比較すると、smfs においては15%程度、snfs においては7%程度のオーバヘッドが測定された。このオーバヘッドは、主にソースコードを変更しないで済ますために必要な関数のコストが原因であると考えられる。

5 関連研究

新たなファイルシステムやデバイスドライバなどのモジュールを組み込んで OS を拡張した際に、OS の安定性を確保する研究は他にもいくつかなされている。しかしどの研究も、全てのモジュールに対して最適な一種類の強さの保護を模索し、提供しようとしているにすぎない。

Mach [6] ではモジュールをユーザプロセスとして実装することができる。それによってモジュールは完全に保護され、カーネルや他のユーザプロセスから切り離されるが、その代わりに実行効率が非常に悪くなっている。

SPIN では型安全な言語である Modula-3 [7] を使ってモジュールを記述する。これによってモジュールの安全性はコンパイル時に静的に検査されるので、実行時にはほとんどオーバヘッドなしにモジュールを実行することができる。ただし Modula-3 は、システム記述言語として最も一般的に使われている C 言語ほど普及しておらず、再利用できる既存のコードもあまりない。

SPIN のように型安全な言語で安全性を保証する代わりに、インタプリタを使って実行時に安全性を検査することもできる [8]。この方法ならばコンパイラが静的に検査できないような不正なアクセスを検出することも可能であるが、実行時のオーバヘッドが非常に大きくなる。

VINO [9] では Software Fault Isolation (SFI) [10] を使ってモジュールを安全にしている。SFI ではメモリへの書き込みやジャンプ命令の前にその正しさを検査するコードを挿入する。そして実行時に検査して、不正な振舞いをしてもそのモジュール以外には影響が及ばないようにしている。SFI は多段階保護機構の弱い保護として利用することができると考えられる。

6 まとめと今後の課題

拡張したことにより OS が不安定になるのをできるだけ避けつつ、実行効率をよくできるように、多段階保護機構を提案した。そしてその実例としてファイルシステムのための多段階保護機構を実装し、実験によってその有用性を示した。

多段階保護機構は OS を拡張するためのモジュールの不正な振舞いから OS を守るためだけではなく、アプリケーションなどのシステム一般に利用することができると考えられる。

本稿ではファイルシステムに限定して多段階保護機構を導入したが、今後はその他のサブシステムに対しても導入していき、より汎用的な実装にしていく予定である。

参考文献

[1] Engler, D. R., Kaashoek, M. F. and Jr., J. O.: Exokernel: an operating system architecture for application-level resource management, in *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, Vol. 29 (1995).

[2] Bershad, B. N., Savage, S., Pardyak, P., Gün Sirer, E., Fiuczynski, M., Becker, D., Eggers, S. and Chambers, C.: Extensibility, Safety and Performance in the SPIN Operating System, Technical report, Department of Compuer Science and Engineering, Univ. of Washington (1995).

[3] McKusick, M. K.: The Virtual Filesystem Interface in 4.4BSD, *Computing Systems*, Vol. 8, No. 1, pp. 3–25 (1995).

[4] Sandberg, R.: Design and Implementation of the Sun Network Filesystem, *Proceedings of the USENIX 1985 Summer Conference* (1985).

[5] Sun Microsystems, Inc., Mountain View, CA: *Remote Procedure Call Protocol Specification* (1986).

[6] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach: a new kernel foundation for UNIX development, *Proceedings of the USENIX 1986 Summer Conference (Atlanta)*, pp. 93–112 (1986).

[7] Nelson, G.: *System Programming with Modula-3*, Innovative Technology, Prentice Hall (1991).

[8] 西村健, 猪原茂和, 益田隆司：ユーザレベルプロトコルのカーネル内実行による大量データ通信の効率的実現, 情報処理学会研究報告 (1996).

[9] Christopher Small, M. S.: VINO: An Integrated Platform for Operating System and Database Research, Technical report, Harvard University (1994).

[10] Wahbe, R., Lucco, S., Anderson, T. E. and Graham, S. L.: Efficient Software-Based Fault Isolation, in Liskov, B. ed., *Proceedings of the 14th Symposium on Operating Systems Principles*, pp. 203–216, New York, NY, USA (1993), ACM Press.