システムソフトウェアと
オペレーティング・システム 73－4
（１９９６．８．２６）

アプリケーション内メモリ管理の枠組みと分散環境における選択的な保護

古川 陽　大西雅之　柴山悦哉
東京工業大学 大学院 情報理工学研究科 数理・計算科学専攻

従来オペレーティングシステムにより行われていたメモリ管理をアプリケーション内で行う枠組みについて述べる．アプリケーション内メモリ管理により，アプリケーションのコンテキスト情報を利用した効率的なページングやアプリケーション間の情報共有形態に応じたアドレス付けを実現できる．さらに，オペレーティングシステム内に隠蔽されていたページングに関する情報などを利用することにより，アプリケーションの実行の最適化を図ることができる．また，アプリケーション制御可能な細粒度保護ドメインの導入により，分散環境において信頼できるプログラムとそうでないものについて選択的な保護を行う方法について検討する．

A Framework for Application Level Memory Management and Selective Protection

Yo FURUKAWA　Masayuki OHNISHI　Etsuya SHIBAYAMA
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

{furukawa,oonisi,etsuya}@is.titech.ac.jp

We propose a framework (i.e., a model and programming interfaces) for application-level memory management, with which applications can exploit information that would be hidden by traditional operating system kernels. Besides, applications can explicitly specify paging, addressing, and protection policies/mechanisms in their memory management modules. Our framework is particularly useful for building a component-based application including network loadable codes via Internet and thus possibly untrusted.

1 はじめに

従来，オペレーティングシステムのメモリ管理機構はカーネル内に固定的に提供されてきた．このようなオペレーティングシステムのサービスは多くのアプリケーションで平均的に良い方策にもとづいて実装されるため，アプリケーションによっては性能低下の原因となることが知られている．このため，ユーザレベルでメモリ管理の方策を変更できる機構がいくつか提案されてきた．しかし，アプリケーションの知識を十分活用した管理方策の実現のためには管理機構とアプリケーションの緊密な連携が必要である．多くのシステムではメモリ管理機構が保護機構と密接に関連しており，カーネルの保護を重視する方針から，必ずしもアプリケーションが十分な知識を得ることができなかったり，アプリケーションの意向を十分に反映できなかった．

一方で，コンポーネントウェアや Java のアプレットなどが新たなアプリケーションの形態として認知され，普及しつつある．複数の開発者によるコンポーネントが組み合わされて動作する環境を想定したとき，それらの間の効率的なデータ共有のためにアドレス空間を共有したり，コンテキスト単位に保護を設定するなど，実行環境に対する多様な要求があると考えられる．このため，アプリケーションコンテキストに即したアドレス空間の構成や保護ドメインの設定など細粒度のメモリ管理が必要となる．

本研究では，カーネルに代わって，アプリケーション内でオペレーティングシステムサービスを実現するための枠組みを提案する．カーネルにより抽象化されたサービスや保護を与えるのではなく，アプリケーション自身が管理することで，プログラマの持つ知識やコンテキストに依存した最適化を図ることができる．アプリケーションはそのコンテキストに応じて自身のアドレス空間構成の変更，ページング機構の操作やメモリの保護を設定できる．これらはアプリケーションの要求に応じて独立に変更できるので，プログラマは必要な機能のカスタマイズだけを行うことができる．

また，実験として Java にアプリケーションから制御可能な細粒度保護ドメインを導入した．この仕組みを用いて，分散環境において信頼できるコンポーネントとそうでないものを選択的に別々の保護ドメイン内で実行することができる．

2 アプリケーション内メモリ管理

オペレーティングシステム内に実装されたメモリ管理部はアプリケーションのアクセスパタンを考慮していないために，アプリケーションによっては極端に性能が低下することが知られている．例えば，画像処理ソフトウェアは画像のフィルタ処理のため短時間のうちに多くのメモリ領域にアクセスするが，これをオペレーティングシステムにより提供される仮想記憶の上に素朴に実装すると極端に低い性能しか得られない．フィルタ処理では近い将来アクセスするページが予測しやすいので，それらをあらかじめメモリにロードすることで大幅に性能を改善できる．

このような問題に対処するために，いくつかのマイクロカーネル構成のオペレーティングシステムではメモリ管理機能を実現する部分とその管理方策を決める部分を分け，管理方策をユーザレベルサーバとして実装することを可能としている．しかし，ユーザレベルサーバによる実装ではいくつか問題がある．

- メモリ管理の機構や方針のうち，どの処理をカーネル内で行ない，どの処理をユーザレベルサーバで行うべきかの切り分けが固定的である．
- ユーザレベルのメモリ管理サーバはカーネルから非同期に送られる要求を処理しなくてはならないので，多くの場合複雑なものとなり，容易に変更したり置き換えることができない．
- カーネルとユーザレベルサーバ間のシステムコールを介した通信は，カーネルモードへのトラップと保護のためのパラメタチェックを伴うため非効率である．

アプリケーションに適したメモリ管理の実現のためには，アプリケーションとメモリ管理部のそれぞれが持つ知識を十分に活用することが重要である．例えば，メモリ管理部が持つ知識としては，

- どのページが物理メモリにロードされているか
- ＴＬＢやページテーブルの内容

などがあり，アプリケーションが持つ知識には，

- 静的に予測できるアプリケーションのアクセスパタン
- 実行時に動的に得られるアクセスタイミング

に関する情報がある．しかし，多くのユーザレベルメモリ管理のシステムでは，アプリケーションに必要な知識がメモリ管理部に隠蔽されており，また，アプリケーションに自分の性質に関する知識をメモリ管理部に伝える手段がない場合や通信に大きなコストがかかることがある．

このようなユーザレベルメモリ管理の難点を解消するため，アプリケーション内でメモリ管理をする研究が行われている [5]．ユーザレベルメモリ管理とアプリケーション内メモリ管理の違いとしては，

- アプリケーション自身のページングやアドレス付けを決定できなど，柔軟にメモリ管理を制御できる．
- メモリ管理に関するすべての情報がアプリケーションから利用できる．

などが挙げられる．以下，アプリケーション内メモリ管理についてアドレス空間構成，保護，ページングの3つの項目を検討する．

2.1 柔軟なアドレス空間の構成

UNIX などプロセスごとに分離したアドレス空間を持つオペレーティングシステムに対して，プロセス間で単一のアドレス空間を共有する方式がいくつか提案されている [2]．単一アドレス空間はプロセス間でポインタを含んだデータ構造の共有が容易であるという利点を持つ一方で，アドレス管理に一定のコストが必要であることや，スケーラビリティがない，UNIX のエミュレーションが難しいなどの難点がある．逆に，UNIX ではアドレス管理コストが少ない反面，細粒度のデータ共有が難しいことや，アドレス空間と保護ドメインの関係が固定的であることが短所として挙げられる．アドレス空間の構成法はそれぞれに得失があり，一般にどの構成法が最適であるかはアプリケーションに依存する．コンポーネントウェアやアプレットなど新しい実行形態の出現により，従来のプロセスという枠組みにとらわれないコンテキスト間の効率的な細粒度のデータ共有が重要となる．そのような状況では，コンテキスト間，保護ドメイン間のアドレス共有や，アプリケーション内の動的なアドレス空間の組み替えなどが必要となる．

2.2 選択的な保護ドメイン

UNIX など従来のオペレーティングシステムでは，保護ドメインはプロセス単位で提供され，プロセス内の各スレッド間に保護はない．しかし，コンポーネントウェアやアプレットなどの利用を考えると，異なる形態の保護ドメインが必要となる．コンポーネントウェアは，複数の異なる開発者によるコンポーネントを組み合わせてアプリケーションとして利用する．そのため，同じアプリケーション内で他人の開発したコンポーネントが予想しない動作をする可能性がある．また，ネットワークから動的にコードを供給するものでは，悪意のあるコードがローカルマシンのデータを破壊してしまうことが考えられる．このような状況では，コンテキストごとの保護など，細粒度の保護機構を提供する必要がある．

2.3 ページングの効率化

従来，ＬＲＵなどオペレーティングシステム内で標準的に実装されているページングアルゴリズムがアプリケーションの実行効率を落としているため，ユーザレベルでアプリケーションに適したページングアルゴリズムを実現する研究が数多く行われている．しかし，ガーベージコレクションやソフトウェアで実現する分散共有メモリなどメモリ管理に密着したシステムの場合，アルゴリズムの変更といった間接的な方法では十分な効率が得られない場合が多い．

例えば，ガーベージコレクションにおいてはアプリケーションとメモリ管理部が連携することによるいくつかの効率化手法が知られている．

- **アプリケーションの知識をメモリ管理部が利用する例**
  コレクタによって既にスキャンされたページはゴミしか含んでいないのでページ管理部はそれをディスクに書き戻す必要はなく，ディスクI/Oを避けることができる [5]．
- **メモリ管理部の知識をアプリケーションが利用する例**
  物理メモリにロードされているページ上のゴミだけを回収することによりページングの回数を減らす [3]．
- **ページングハードウェアをアプリケーション内メモリ管理の効率的な実現のために利用する例**
  実時間ガーベージコレクションではアプリケーションとコレクタが並行に動作するため，現在回収中のページにアクセスするのを防ぐ必要があるが，ページのアクセス権を変更することによりアクセスを検知する方法がある [2]．

ソフトウェア分散共有メモリの実装では，分散共有メモリサーバにリモートアクセスのオーバーヘッドを低減するためページのキャッシュを設け，キャッシュの一貫性管理のためにページングハードウェアを用いるのが一般的である．しかし，キャッシュの効率的な管理やアクセスレイテンシの低減のためにはアプリケーションとサーバ間で必要なページの情報を頻繁に交換する必要がある．このため，アプリケーション内に分散共有メモリサーバの役割を実装したほうが効率化が望める．

3 メモリ管理モジュールの開放的な実装

本研究では，アプリケーション内でメモリ管理機能を実現する枠組みを提案する．すなわち，カーネルにより提供されるメモリ管理機能をアプリケーションが使用するのではなく，アプリケーション内にそのメモリ管理機能を実装する手法をとる．アプリケーションはそのコンテキストに応じて自身のアドレス空間構成の変更，ページング機構の操作やメモリの保護を設定できる．なお本稿では，アプリケーションとしてミドルウェアや言語処理系のランタイム，オペレーティングシステムサブシステムなどを主として想定している．

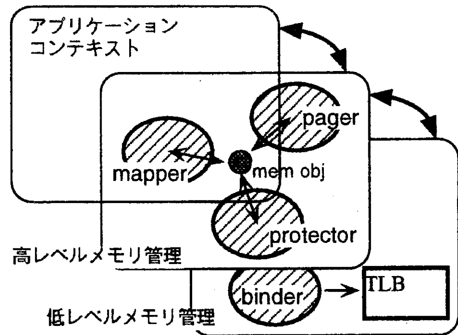
3.1 アプリケーション内メモリ管理の枠組み

提供される枠組みは，メモリ管理モジュールを記述するためのオブジェクトとプロトコルを表す抽象モデル，それを実現する置き換え可能なデフォルトライブラリ，アプリケーションスケルトンからなる．ここでは，アプリケーション内メモリ管理のための抽象モデルについて述べる．メモリ管理モジュールはメモリオブジェクトを操作する3つの高レベルモジュールとそれらを結合する1つの低レベルモジュールからなる．高レベルメモリ管理層では，メモリオブジェクトを中心とした束縛の操作により，ページング，アドレス付け，保護の方策を実装する．各オブジェクトはケイパビリティを用いて管理される．低レベルメモリ管理層では，高レベル管理層の要求をＴＬＢやページテーブル操作に変換する．プログラマは必要に応じてメモリ管理機構の一部を独立に変更できる．

通常，アプリケーション自身にメモリ管理の権限を与えた場合，アプリケーションやカーネルの保護は得られない．ここでは，メモリ管理の方策を決定する層と実際の操作に変換する層に分けることにより，プログラムの安全性を確保する．プログラマは必要に応じてメモリ管理モジュールを段階的にカスタマイズすることができる．高レベル管理層では，割り当てられた物理資源のみについて管理方策を決定できる．物理資源はケイパビリティで指定され，低レベル管理層に束縛を要求される．低レベル管理層におけるチェック機構により，不正な操作や破壊的な操作は検知でき，高レベル管理層におけるプログラムの誤りが他のアプリケーションに影響を与えないことが保証できる．

- **mapper**
  メモリオブジェクトとアドレス領域の束縛を定義する．メモリオブジェクトはアプリケーションが読み書きする対象であり，アドレス領域はメモリオブジェクトに与えられる名前である．アプリケーションはメモリオブジェクトにアドレス領域を束縛することでアドレス付けを行うことができる．この束縛の集合がアプリケーションのアドレス空間となる．

  共有アドレス空間を構成する場合は，アドレス空間を共有する他のアプリケーションの mapper と協調する仕組みを実装する．UNIX のような分離アドレス空間を実装するためには他のアプリケーションとの連携は不要である．



- **pager**
  メモリオブジェクトと物理ページの束縛を定義する．ページングはメモリオブジェクトと物理ページの束縛を操作することで行う．メモリオブジェクトの永続的記憶管理は別に実装される．ＴＬＢミスが起こると，ページフォルトを binder がハンドルし，該当するメモリオブジェクトとともに pager に通知される．pager はメモリオブジェクトと物理ページのケイパビリティを提示し，binder にＴＬＢエントリの作成を依頼する．

- **protector**
  メモリオブジェクトと保護属性の束縛を定義する．コンテキストに応じて読み込み専用や読み書き不可にするなどの属性を変更することにより細粒度の保護を実現できる．

  従来のスレッド間の相互の信頼度に関する情報を持たないオペレーティングシステムではすべてを信用するかすべてを信頼しないかの選択であった．カーネル保護のためにページテーブルやＴＬＢエントリは保護されていなければならないため，通常のオペレーティングシステムではユーザレベルプロセスはエントリを操作できず，ＴＬＢ操作はメモリ管理部の特権的な操作である．このため，アプリケーション内で保護

ドメインを細粒度制御することは不可能であった．このモデルでは protector をカスタマイズすることにより，アプリケーションが保護ドメインのサイズや構造，保護のレベルを自由に変更できる．

- **binder**
  mapper, pager, protector の定義する束縛を統合してＴＬＢの操作に変換する．binder はＴＬＢエントリを操作するための保護されたインタフェースを提供する．このインタフェースの保護レベルはアプリケーションプログラマが変更することもできる．デフォルトで提供されるインタフェースではケイパビリティを検証することで操作の正当性を確認する．アプリケーションプログラマはこのチェックを省くことでより低コストで細粒度の操作を実装することもできる．実行時の保護レベルはユーザやシステム管理者がこれらのインタフェースを選択することで制御できる．

4 利用例：軽量保護ドメインの導入

今回，アプリケーション内メモリ管理のアドレス付け，ページング，保護の３つの機能のうち，アプリケーションレベルの保護管理の応用例として，Java にプログラムから制御可能な細粒度保護ドメインを導入した．Java はネットワークを通じてプログラム (クラスファイル) をネットワークから動的にロード出来ることを一つの特徴とする言語である．インターネット環境では接続先が必ずしも信頼できる相手とは限らないので，ロードしたクラスファイルに対する十分な認証と、不正な動作に対する防御が重要である．Java ではクラスファイルを実行前に検証し，

- 正常に実行できるプログラムであるか
- 不正なメモリ参照はないか
- ランタイムのスタックを破壊しないか

などを調べている．しかし，これは主としてランタイム処理系の動作に支障をきたさないことを保証するためのものであり，必ずしもユーザの意図しない動作を起こさないことを保証できるものではない．そこで，Java に細粒度の保護ドメインの機能を導入して，スレッドごとに参照可能なメモリ領域を管理することを考える．ロードしたクラスファイルのコードを実行する前にスレッドを分岐して，異なる保護ドメインで実行すれば，そのコードが意図しないメモリ参照を起こすものであったとしても，その動作を検知することができる．

また，このオペレーティングシステムの保護に相当するこの仕組みを利用すれば，Java による大規模なプログラムの効率的な開発を支援できる．複数のスレッドが並行動作するプログラムの開発では，各スレッドごとにアクセスできる領域を限定することにより，意図しないメモリ参照が検知ができ，効率的にデバッグを行うことができる．

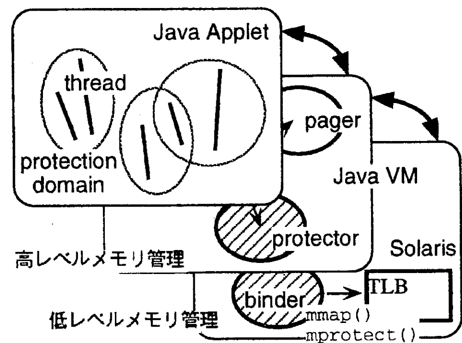
4.1 保護ドメインクラス

Java プログラムから細粒度保護ドメインを生成するインタフェースをクラスの形で提供する．

```
public class ProtectionDomain {
...
  // 保護ドメインに加える
  public static native void join(Thread th);
  // 保護ドメインから外す
  public static native void leave(Thread th);
...
}
```

生成した保護ドメインにスレッドを動的に追加，削除することで同じドメインに属するスレッドのグループを形成することができる．保護ドメインに属さないスレッドからのメモリ参照には例外が発生し，適切なエラー処理を行なうことができる．

4.2 JavaVM に対する変更



上記のような細粒度保護ドメインの有効性を検討するために，Solaris 上の JavaVM に変更を加える形でプロトタイプシステムを実現した．プロトタイプシステムでは，JavaVM のスレッドごとのスタックやヒープ領域をメモリオブジェクトとして，protector でその保護属性を管理し，binder の TLB 操作を Solaris の mprotect() システムコールで代用した．スレッドのコ

ンテキストスイッチのたびに，必要ならば，mprotect() システムコールを用いスタックやヒープ領域に対するアクセス権を変更することで保護ドメインを実装した．また，JavaVMの命令インタプリタによって動的に行われているチェックの一部をページングハードウェアを用いて実現することにより，実行を効率化できる．

5 関連研究

exokernel[4, 5] は管理する資源の多重化のみを行ない，一切の抽象モデルを提供しないオペレーティングシステムである．物理メモリやＴＬＢを仮想メモリという形で抽象化するのではなく，直接的な形でアプリケーションに受け渡す．この方式は，柔軟性と効率の面からは魅力的であるが，移植性がない，わずかな変更をするためにも物理資源の知識が必要となるなどの欠点がある．本研究は，同様の柔軟性を保ったまま，メモリ管理のためのプログラミングモデルを与え，アプリケーションプログラマの負担を軽減している．

拡張可能な資源管理の例として *SPIN*[1, 6] が挙げられる．*SPIN*では，低レベル資源をアプリケーションレベルで安全に管理するための拡張可能なインタフェースを提供している．ユーザが記述した拡張は効率と細粒度制御のためにカーネルにダイナミックリンクされる．この拡張からカーネルを守るために型安全な言語とリンク時のチェック機構を利用している．ユーザはアプリケーション内で割り当てられた物理ページと仮想アドレス領域の間の束縛を要求して，アドレス付けとページングを制御することができる．これらのオブジェクトおよび束縛はケイパビリティベースのシステムにより保護されている．しかし，この方式では，ページングだけを操作したい場合にもアドレス付けを考慮しなければならないなどプログラマの負担は大きい．本研究では，ページングとアドレス付けを独立に操作でき，必要に応じたカスタマイズを可能にしている．

6 まとめ

本稿では，アプリケーション自身によるメモリ管理の有用性について検討し，アプリケーション内メモリ管理のの枠組みを提案した．この枠組みを用いれば，プログラマはアプリケーションの特質に応じて，アドレス付け，ページング，保護ドメインを独立にカスタマイズすることができる．また，アプリケーションのコンテキストに応じてそれらの動的な最適化を行うことが可能となる．今後は，さまざまな実験を通じて，その有用性と効率を評価する予定である．

参考文献

[1] Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc Fiuczynski, David Becker, and Susan Eggers Craig Chambers. Extensibility, Safety and Performance in the SPIN Operating System. Technical report, University of Washington, 1995.

[2] J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska. Sharing and Protection in a Single Address Space Operating System. *ACM Transactions on Computer Systems*, 1994.

[3] Alan Demers, Mark Weiser, Barry Hayes, Hans Boehm, Daniel Bobrow, and Scott Shenker. Combining Generational and Conservative Garbage Collection: Framework and Implementations. In *17th Annual ACM Symposium on Principles of Programming Languages*. ACM, 1990.

[4] D. R. Engler, M. F. Kaashoek, and Jr. J. W. O'Toole. The Operating System Kernel as a Secure Programmable Machine. *OPERATING SYSTEMS REVIEW*, Vol. 29, No. 1, pp. 78 – 82, January 1995.

[5] Dawson R. Engler, Sandeep K. Gupta, and M. Frans Kaashoek. AVM: Application-Level Virtual Memory. Technical report, MIT Laboratory for Computer Science, 1995.

[6] Stefan Savage and Brian N. Bershad. Some Issues in the Design of an Extensible Operating System. In *First USENIX Symposium on Operating Systems Design and Implementation*, p. 196. ACM, November 1994.

[7] Sun Microsystems, Inc. Java: Programming for the Internet, 1995. http://java.sun.com/.