キー/ロック方式の拡張によるアクセスルートコントロール

繁田聡一[†] 清水謙多郎[††] 曽和将容[†]

新たなサービスを提供するオブジェクトを動的に追加することで機能拡張ができるシステムソフトウェアは，サービスを提供する複数のオブジェクトが動的にロード・リンクされ，互いに協調して動作することで実現される．ここで，セキュアな拡張可能システムを実現するためには，協調関係にないオブジェクトを実行中のサブジェクトからの不正なアクセスを防止しなければならない．そこで我々は，複数のオブジェクトの多様な協調関係に柔軟に適応できる新しいアクセス制御機構として拡張キー/ロック方式を開発した．拡張キー/ロック方式では，サブジェクトは，オブジェクトを呼び出したときに動的にキーを獲得し，これがオブジェクトへのアクセス履歴として利用される．また，オブジェクトの持つロックリストには，複数のキーを同時に使用しなければ開錠できないロックを登録することができる．この2つの拡張によって，従来のアクセス制御機構では提供できなかった，サブジェクトが協調関係にあるオブジェクトを経由してきたか否かに基づいたアクセス制御を提供する．我々は，これをアクセスルートコントロールと呼ぶ．本論文では，拡張キー/ロック方式の設計，拡張可能なオペレーティングシステムへの適用，性能に関する考察について述べる．

Access Route Control Based on the Key/Lock Scheme

Soichi Shigeta,[†] Kentaro Shimizu[††] and Masahiro Sowa[†]

Extensible system software is constructed as a set of many fine-grained objects that can be linked and loaded dynamically. The services of such system software are often implemented by the cooperation of multiple objects that are invoked from each other. Conventional access control schemes are not flexible enough to protect such services. Therefore we developed a new flexible, fine-grained access control sheme, called extended key/lock scheme. In the extended key/lock scheme, subjects acquire keys from objects when the object invocation is successful. These keys represent subjects' history of object invocation. Users can specify a lock that is opened with combination of multiple keys in object's lock list. These extension realize access route control, that is, only subjects that invoking particular set of cooperative objects are permitted to access the object. In this paper, we describe the design of extended key/lock scheme, integration to an extensible operating system, and cost evaluation.

1. はじめに

種々のハードウェアプラットフォームやユーザアプリケーションからの様々な要求に対応するために，オペレーティングシステムを拡張可能なソフトウェアとして設計することが重要となっている．互いに協調して動作する細粒度なモジュールの集合としてオペレーティングシステムを構成すること[1),6)]で拡張性が実現されている．協調する複数の保護されたモジュールが要求に応じて同じアドレス空間に動的にリンク，ローディングされなければならないため，個々のアドレス空間に対応する既存の粗粒度なモジュール（たとえば，UNIX のプロセス）で実現することはできない．モジュールの動的ローディングや動的リンク，モジュール間でのスレッド移送といった，拡張可能なシステムを構成するための様々な基礎技術はすでに開発されているが，拡張可能なオペレーティングシステムに適した保護機構については現在さかんに研究が行われている[2),5)]．細粒度なモジュールで構成される拡張可能なオペレーティングシステムの設計においては，1つのアドレス空間内でモジュールを不正なアクセスから保護するための機構が重要な鍵となる．

ユーザにオペレーティングシステムのサービスを拡張することを許すためには，ユーザが必要に応じてサービスモジュールを追加したり置き換えたりできるようにしなければならない．特に，CPU スケジュー

† 電気通信大学情報システム学研究科
Graduate School of Information Systems, University of Electro-Communications
†† 東京大学農学生命科学研究科
Graduate School of Agricultural and Life Sciences, the University of Tokyo

リングやページ置換え，ディスクスペース割当てなどといった資源管理ポリシーを動的に置き換えて利用することができれば，個々のアプリケーションに適した効率的な資源管理を実現することができる．しかしながら，ユーザによるサービスモジュールの実装はシステムのクラッシュや他のサービスの不正な使用を引き起こす可能性があるため，すべてのモジュールは互いに保護されていなければならない．すなわち，互いに協調関係にあるモジュールを実行中のスレッドだけにアクセスが許可されるように制御する機構が必要である．そこで，そのようなアクセス制御を実現する拡張キー/ロック方式[13),14)]を開発した．アクセスを行う主体をサブジェクト，アクセスされる対象をオブジェクトという一般的な用語で表現すると，拡張キー/ロック方式は次のような特徴を備えている．

(1) 1つのアドレス空間内に複数の保護ドメインを定義することができる．
(2) オブジェクトへのアクセス権は個々のサブジェクトごとに異なり，サブジェクトのアクセス権はオブジェクト呼出しごとに動的に変化する．
(3) オブジェクトへのアクセス条件として，協調関係にあるオブジェクトの呼出し系列を経たか否かという，サブジェクトのアクセス履歴を設定することができる．
(4) 階層的，統合的なアクセス権の設定を支援する，サブジェクトやオブジェクトをグルーピングする機能を提供する．

拡張キー/ロック方式そのものは，分散オブジェクトやモバイルオブジェクトのアクセス制御にも有効であると考えられるが，本論文では，単一アドレス空間方式の拡張可能なオペレーティングシステムへの適用について論じる．

2. 従来のアクセス制御方式

本章では，従来のアクセス制御方式について説明し，次章で我々が開発した拡張キー/ロック方式について述べる．アクセスを行う主体をサブジェクト，アクセスされる対象をオブジェクトと呼ぶ．アクセス制御機構とは，アクセス権を与えられたサブジェクトだけがオブジェクトに対する操作を許可されるように制御する機構である．すなわち，オブジェクトを不正なアクセスから保護するための機構である．

オペレーティングシステムで用いられているアクセス制御方式は，ケイパビリティ方式かアクセス制御リスト方式のいずれかが一般的である．これに対して，本論文で述べるアクセス制御方式はキー/ロック方式を拡張したもので，オブジェクトに対する操作を許されるサブジェクトの条件を柔軟に設定できるという特徴がある．

2.1 ケイパビリティ方式

サブジェクト Sにケイパビリティ$\mathrm{C}(O, R)$（O：オブジェクトの識別子，R：操作の集合）を配布する．サブジェクト Sがオブジェクト Oに対する操作 rを許されるのは，$\mathrm{C}(O, R)$; $r \in R$ の条件を満たすケイパビリティを持っている場合だけである．

- 長所：実行時にケイパビリティを動的に獲得することで，サブジェクトは動的にアクセス権を獲得することができる．また，権限のあるサブジェクトはケイパビリティを他のサブジェクトに譲渡して，アクセス権を与えることができる．
- 短所：アクセス権が記述されたケイパビリティをサブジェクトに配布してしまうため，アクセス権の無効化や変更が困難である．

2.2 アクセス制御リスト方式

ケイパビリティ方式とは逆に，オブジェクト Oに(S, R)（S：サブジェクトの識別子，R：操作の集合）の組をリスト形式で持たせる．これをアクセス制御リスト $\mathrm{ACL}(O)$ という．サブジェクト Sがオブジェクト Oに対する操作 rを許されるのは，$\exists(S, R) \in \mathrm{ACL}(O)$; $r \in R$ の条件が満たされる場合だけである．

- 長所：オブジェクトに対するアクセス権が，そのオブジェクトの持つアクセス制御リストで管理されているため，アクセス権の無効化や変更はそれを修正することで容易に行うことができる．
- 短所：アクセス権をサブジェクト間で譲渡することができない．

2.3 キー/ロック方式

サブジェクト Sは配布されたキー kをキーリスト $\mathrm{KL}(S)$ に保持する．オブジェクト Oには (l, R)（l：ロック，R：操作の集合）の組をリスト形式で持たせる．これをロックリスト $\mathrm{LL}(O)$ という．サブジェクト Sがオブジェクト Oに対する操作 rを許されるのは，$\exists k \in \mathrm{KL}(S) \wedge \exists(l, R) \in \mathrm{LL}(O)$; $l = k$, $r \in R$ の条件が満たされる場合だけである．

キー/ロック方式は，ケイパビリティ方式とアクセス制御リスト方式を一般化した方式と見ることができる．サブジェクトにキーを配布するという点がケイパビリティ方式に，オブジェクトのロックリストにアクセス権が保持される点がアクセス制御リスト方式に類似している．

- 長所：
  - 動的にキーを獲得することで，サブジェクト

は動的にアクセス権を獲得することができる．また，権限のあるサブジェクトは他のサブジェクトにキーを譲渡して，アクセス権を与えることができる．
  - オブジェクトの持つロックリストで，そのオブジェクトに対するアクセス権が管理されているため，それを修正することでアクセス権の無効化や変更が容易に行える．
- 短所：キーリストとロックリストという2つのリストを参照して，キーとロックのマッチングを調べなければならないため，他のアクセス制御方式に比べてアクセス権チェックのコストが大きい．

3. 拡張キー/ロック方式

我々が開発したアクセス制御機構は，従来のキー/ロック方式を拡張したものである．これを拡張キー/ロック方式と呼ぶ．本章では，拡張により追加された機能と，その機能を利用して実現されるアクセスルートコントロールについて述べる．以下の議論では，オブジェクトはプログラムのコードおよびデータを提供する受動的な実体，サブジェクトはコードを実行する能動的な実体，すなわち，スレッドを仮定する．複数のキーの組合せでアクセス条件を記述できる拡張キー/ロック方式そのものは，従来のキー/ロック方式と同様に一般的なアクセス制御に適用することができるが，本論文では，提案方式を単一仮想アドレス空間方式の拡張可能なオペレーティングシステムに適用することを念頭に，このような仮定を行った．

3.1 拡張された機能

複数のオブジェクト間の協調関係を記述する機能とサブジェクトのアクセス履歴を表現する機能を提供するために，次のような拡張を行った．

- 各サブジェクト Sは，キーリスト $\mathrm{KL}(S)$ を保持する．サブジェクトのアクセス権は，このキーリストに含まれるキーによって規定される．一方，各オブジェクトもオブジェクトキーリスト $\mathrm{OKL}(O)$ を保持する．このオブジェクトキーリストのキーは，アクセス権チェックを実現するモジュール（たとえば，4.6 節で後述するアクセス権チェックハンドラ）によってオブジェクトの呼出しを許可されたサブジェクトに継承される．このとき，サブジェクトの保護ドメインは，呼び出したオブジェクトによって規定される保護ドメインに暗黙的に移る．サブジェクトがオブジェクト呼出しから復帰するときには，サブジェクトがオブジェクトから継承したキーは没収される．
- 従来のキー/ロック方式ではキーとロックの対応関係は1対1であるが，拡張キー/ロック方式では複数のキーの同時使用によって開錠されるロックを設定できる．呼び出したオブジェクトからサブジェクトがキーを獲得するので，特定のオブジェクト呼出し系列（これをアクセスルートと呼ぶ）を経ていることをアクセス条件として指定することができる．
- キーのグルーピングを行うことができる．キーグループは，サブジェクトやオブジェクトの階層的，統合的な管理やアクセス制御を提供する．

拡張キー/ロック方式では，オブジェクトのロックリスト $\mathrm{LL}(O)$ に，複数のキーを同時に使用しなければ開錠することのできないロック（$l = k_1 \wedge k_2 \wedge \cdots \wedge k_n$）を作成することを許す．サブジェクト Sにオブジェクト O に対する操作 rが許されるのは，$\exists k_1, \exists k_2, \cdots, \exists k_n \in \mathrm{KL}(S) \wedge \exists (l, R) \in \mathrm{LL}(O);\ l = k_1 \wedge k_2 \wedge \cdots \wedge k_n,\ r \in R$ の条件が満たされる場合だけである．この拡張が，アクセスが許される条件を柔軟に指定することを可能とする．また，サブジェクトの持つキーリスト $\mathrm{KL}(S)$ には，生成時にシステムから与えられるキーだけではなく，呼び出したオブジェクトから継承されるキーが保持される．これによって，サブジェクトの実行の流れに従ってキーリストに含まれるキーの集合が変化し，サブジェクトの実行履歴の情報をアクセス制御の情報として用いることができる．

3.2 キーリストとアクセス履歴

図1 に，サブジェクトのキーリストにオブジェクトへのアクセス履歴が保持される様子を示す．オブジェクトを呼び出したときに，サブジェクトはオブジェクトの持つオブジェクトキーリストに登録されているキーを獲得する．図1では，サブジェクト Sがオブジェクト Aからオブジェクト Bを呼び出したときに，

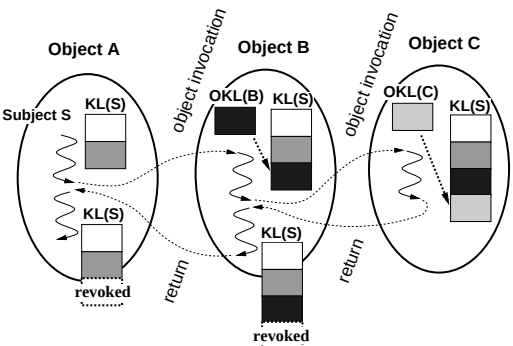

図1 モジュール呼出しにともなうキーリストの変化
Fig. 1 Object invocation and key list management.

B のオブジェクトキーリスト OKL(B) のキーがサブジェクト S のキーリスト KL(S) に追加されている．同様に，オブジェクト B からオブジェクト C を呼び出したときに，OKL(C) のキーが KL(S) に追加されている．逆に，サブジェクトがオブジェクト呼出しから復帰する際には，オブジェクトから獲得したキーはサブジェクトのキーリスト KL(S) から没収される．図 1 では，たとえば，サブジェクト S がオブジェクト C からオブジェクト B へ復帰した時点では，S が C のオブジェクトキーリスト OKL(C) から獲得したキーは，S のキーリスト KL(S) から没収されている．

したがって，サブジェクトのキーリスト KL(S) には，呼び出し中のオブジェクトへのアクセス履歴が保持されている．従来のアクセス制御機構は，(ユーザ id やグループ id といった) サブジェクトの静的な属性のみに基づいていたが，拡張キー/ロック方式は，アクセス履歴というサブジェクトの動的な属性をアクセス制御に組み込んでいる．

3.3 柔軟なロック

論理演算子を用いて複合的なアクセス条件を指定したり，明示的なアクセス拒否を設定したりできる機能を備えた，柔軟なロックを提供する．各オブジェクト O は，ロックリスト LL(O) を持つ．ロックリストの個々のエントリは次の 3 つの要素で構成される．

- ロック
- 操作の集合
- 呼出しに関する指定（grant/deny）

ロックはキーの論理式として記述される．ユーザはキー間に論理演算子 AND，OR，NOT を適用して様々なアクセス条件を柔軟に設定することができる．たとえば，(K_a AND K_b) というロックが指定されたオブジェクトにアクセスするには，サブジェクトはキー K_a と K_b の両者を獲得していなければならない．一方，(K_a OR K_b) というロックが指定されたオブジェクトに対しては，K_a か K_b のいずれかを獲得していればアクセスが許可される．また，(NOT K_a) というロックが指定されたオブジェクトへは，キー K_a を持たないサブジェクトにアクセスが許可される．

操作の集合には，ロックに適合するキーの組合せを持つサブジェクトに対して許可（もしくは禁止）する操作を指定する．

呼出しに関する指定は，操作の集合で指定した操作でオブジェクトにアクセスすることを許可する（grant）か拒否する（deny）かを指定するためのものである．

3.4 アクセスルートコントロール

本論文では，協調関係にあるオブジェクトどうしの一連の呼出し系列を「アクセスルート」と呼び，サブジェクトが特定の「アクセスルート」を経ているか否かによってオブジェクトへのアクセスを制御することを「アクセスルートコントロール」と呼ぶ．

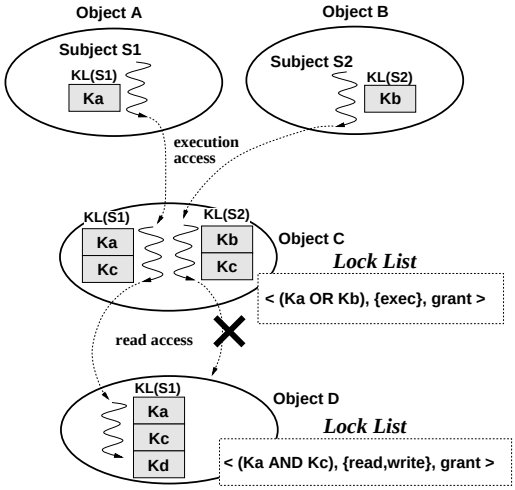

図 2 アクセスルートの指定
Fig. 2 Access route control.

図 2 に，アクセスルートコントロールの例を示す．2 つのサブジェクト *S1* と *S2* がある．オブジェクト A が提供するコードを実行している *S1* のキーリストにはキー K_a が含まれている．K_a がキーリストに含まれていることが，*S1* が A にアクセス中であることを示している．同様に，*S2* のキーリストにはキー K_b が含まれている．オブジェクト C のロックリストには < (K_a OR K_b), {exec}, grant > というロックリストエントリが設定されている（3 つの要素は，3.3 節で述べた各要素に対応する）．これは，キー K_a か K_b のいずれかのキーを持っているサブジェクトに対して，このオブジェクトのコードの実行を許可するという設定である．したがって，サブジェクト *S1* と *S2* の両者ともオブジェクト C を呼び出して使用することができる．3.1 節で述べたように，サブジェクトが C を呼び出したときに，サブジェクトのキーリストにキー K_c が追加される．続いて，両サブジェクトは，C によって提供されるコードに従ってオブジェクト D へアクセスしようとする．ここで，D のロックリストには < (K_a AND K_c), {read, write}, grant > というロックリストエントリが設定されている．それゆえ，2 つのキー K_a と K_c の両者を持っているサブジェクトでなければ D へのアクセスは許されない．この例では，サブジェクト *S1* だけがオブジェクト D に対する読出しと書込みの操作ができる．*S2* はキー K_a を

獲得していない，すなわち，オブジェクト A へのアクセス履歴がないため，D へのアクセスは拒否される．

図 2 の例では，オブジェクト D へのアクセスルートが左側の 1 つのルートに限定されている．システム内に存在するどのオブジェクトも D へのアクセスを試みるコードをサブジェクトに提供しうるため，誤りや悪意による不正なアクセスを防止するには，アクセスルートを制御する機構が必要である．拡張キー/ロック方式では，サブジェクトがオブジェクを呼び出したときにキーを動的に獲得し，復帰時にはキーが没収される機能によって，サブジェクトのアクセス履歴，すなわち，サブジェクトが協調関係にあるオブジェクトを使用中であることが表現されている．また，協調関係にあるオブジェクトから継承する複数のキーを同時に使用しなければ開錠できないロックをロックリストに設定できる機能によって，協調関係に基づいたアクセスルートの指定が実現されている．

4. 拡張可能なオペレーティングシステムへの適用

本章では，拡張キー/ロック方式を単一アドレス空間方式の拡張可能なオペレーティングシステムのアクセス制御機構として適用する．仮想アドレス空間を使用する際の割当てをセグメント（仮想アドレス空間上の連続ページ）単位で行い，プログラムの実行主体であるスレッドが行うセグメントへのアクセスを拡張キー/ロック方式により制御することを検討する．

4.1 拡張可能なオペレーティングシステム

図 3 に示すように，小さなナノカーネルと協調して働く細粒度のモジュールとで構成される，動的に機能拡張可能なオペレーティングシステムを想定する．これらのモジュールは，単一のアドレス空間内で互いに保護されなければならない．ナノカーネルは，スレッド管理やアクセス制御，モジュールの動的バインディングといった基本機能のみを提供する．従来のオペレーティングシステムではカーネルのサービスとして実装されていた，CPU スケジューリングや仮想ページ管理，ディスク割当てといったサービスはナノカーネル外のモジュールとして実装される．モジュールが単一のアドレス空間内にロードされ実行されるため，データ共有やモジュール呼出しを効率的に行うことができる．このような拡張可能なオペレーティングシステム（もしくはオペレーティング環境）は，既存のファイルシステムにトランザクション処理などのより高水準の抽象概念を提供するサービスを追加したり，アプリケーションに特定の資源管理ポリシーを必要に応じて置き換えたりするのに有用である．この特徴は，実時間処理や連続メディア処理を行うシステムで特に有効であることが期待できる．

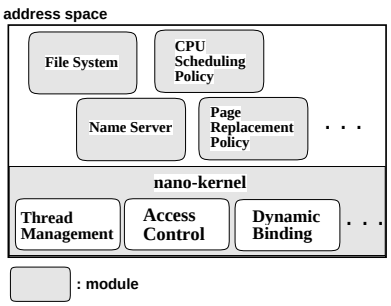

図 3 ナノカーネルと細粒度モジュールにより構成されるオペレーティングシステム
Fig. 3 Operating system constructed as a set of nano-kernel and fine-grained modules.

4.2 サブジェクトとオブジェクト

想定しているオペレーティングシステムでは，唯一の実行主体であるスレッドがサブジェクトに，単一仮想アドレス空間上の連続ページ領域であるセグメントがオブジェクトに対応する．各スレッドはキーリスト KL(*thr*) を持ち，一方，各セグメントにはロックリスト LL(*seg*) が結び付けられている．セグメントのロックリストに登録されているロックのいずれかを開錠できるキーを持つスレッドだけが，そのセグメントへのアクセスを許可される．また，各セグメントにはオブジェクトキーリスト OKL(*seg*) も結び付けられている．OKL(*seg*) に登録されているキーは，そのセグメントを呼び出したスレッドに継承される．

4.3 実行モデル

スレッドは単一アドレス空間内にロードされた複数のテキストセグメントを渡り歩きながら走行する．すなわち，協調関係にあるプログラムモジュールの呼出しと復帰を繰り返しながら走行する．また，スタックセグメントやデータセグメントの参照を行う．3.1 節で述べたように，スレッドがテキストセグメントを呼び出したときに，そのセグメントから動的にキーを獲得する．獲得したキーは，引き続くセグメント呼出しに使用することができる．セグメント呼出しからの復帰時には，それらのキーはスレッドのキーリストから没収される．

4.4 キーとロック

ユーザ識別子やスレッド識別子，セグメント識別子などすべての識別子はキーとして表現される．次の 4 種類のキーを使用する．

- ユーザキー

各ユーザに固有に与えられるキーであり，ユーザ識別子の役割をする．

- スレッドキー
  各スレッドに固有に与えられるキーであり，スレッド識別子の役割をする．
- セグメントキー
  各セグメントに固有に与えられるキーであり，セグメント識別子の役割をする．
- ユーザ定義キー
  ユーザがユーザグループを作成したり，スレッドやセグメントのグルーピングを行ったりする場合に明示的に生成して使用するためのキーである．ユーザが明示的に生成して利用することができるのは，このユーザ定義キーだけである．

セグメント seg のロックリスト LL(seg) には，複数のキーの同時使用によって開錠できるロック $l = k_1 \wedge k_2 \wedge \cdots \wedge k_n$ を設定することができる．この場合，セグメント seg に対する操作 r を許可されるのは，$\exists k_1, \exists k_2, \cdots, \exists k_n \in \mathrm{KL}(thr) \wedge \exists (l, R) \in \mathrm{LL}(O)$; $l = k_1 \wedge k_2 \wedge \cdots \wedge k_n$，$r \in R$ を満たすスレッドだけである．

各スレッド thr のキーリスト KL(thr) には，初期状態では，スレッドのオーナーであるユーザのユーザキーとスレッド識別子となるスレッドキーが含まれている．オーナーを表現するユーザキーには，スレッドをアクティベートしたユーザがシステムにログインしたときに与えられる，システムよって認証されたユーザのユーザキーが用いられる．この 2 つのキーをスレッドのキーリストから削除したり変更したりすることはできない．一方，各セグメント seg のオブジェクトキーリスト OKL(seg) には，初期状態では，セグメントの識別子となるセグメントキーだけが含まれている．このキーをオブジェクトキーリストから削除したり変更したりすることはできない．オブジェクトキーリストに登録されているキーは，そのオブジェクトを呼び出したスレッドに継承される．セグメントのオーナーは，オブジェクトキーリストにユーザ定義キーを追加・削除することができる．

4.5 アクセスルートコントロールの例

たとえば，オペレーティングシステムが管理するスレッド制御情報の格納されたデータセグメントは，その情報に対応する適切なスレッドがスレッドスケジューリングポリシーモジュールを経由した場合にだけアクセス可能でなければならない．3.3 節の図 2 において，オブジェクト A，B をアプリケーションプログラムのテキストセグメント，オブジェクト C をスレッドスケジューリングポリシーを提供するモジュールのテキストセグメント，また，オブジェクト D をオペレーティングシステムがアプリケーション A を実行中のスレッドを管理するためのデータセグメントと対応づける．すると，アプリケーション A を実行中のスレッドがスレッドスケジューリングポリシーモジュール（オブジェクト C）を呼び出している場合にだけ，スレッド制御情報を管理するデータセグメント（オブジェクト D）へのアクセスが許可される．スレッドスケジューリングポリシーモジュールを介さずに，スレッド制御情報にアクセスすることはできない．さらに，たとえスレッドスケジューリングポリシーモジュールを経由していても，制御情報に対応するスレッドでなければスレッド制御情報にアクセスすることはできない．

また別の例として，あるユーザがトランザクション処理を提供するモジュール（トランザクションマネージャ TM）を追加して既存のストレージシステムを拡張することを考える．このとき，トランザクション処理に関するデータオブジェクト DO は，トランザクション処理を必要とする特定のアプリケーションプログラム TA を実行中のスレッドが，トランザクションマネージャを呼び出している場合にのみアクセスされるべきである．したがって，トランザクション処理を必要とする特定のアプリケーションのキー K_{TA} とトランザクションマネージャのキー K_{TM} の両方を持つスレッドだけがアクセスを許可されるように，データオブジェクト DO のロックリスト LL(DO) に $<(K_{TA}$ AND $K_{TM})$, {read}, grant $>$ というロックリストエントリを設定すればよい．さらに，データオブジェクトのオーナーである特定のユーザ foo だけがデータの変更をできるようにするには，ユーザ foo のキーである K_{foo} を追加して，$<(K_{foo}$ AND K_{TA} AND $K_{TM})$, {write}, grant $>$ というロックリストエントリを設定すればよい．このように，拡張キー/ロック方式は，ユーザに理解しやすい形でアクセスルートコントロールを実現することができる．

本節では，オペレーティングシステムのサービスの保護を実現する基盤として，仮想アドレス空間のセグメントを対象としたキー/ロック方式の適用例を示した．セグメントは，1 つまたは複数のページから構成されるものとする．操作の種類としては，コードを格納したセグメントに対しては execute を，それ以外のセグメントに対しては read，write を仮定している．実際のオブジェクトに対する操作には，様々なものが考えられるが，既存の仮想記憶機構への効率的な適用を考え，このような基本的な操作を対象とする（たとえ

ば，コードを格納したセグメントについては，executeの権限により，そこに格納したすべての手続きの実行を認める）．セグメントの生成や破棄といった，その他の操作に対しても保護を適用しようとする場合は，アクセス権を表現するためのビット数が増加するが，現実の利用では32ビット程度に抑えることができると考えられる．その場合，アクセス権チェックのコストの増加はわずかである．

4.6 TLBを用いたアクセス権のチェック

以下では，MIPS R10000やCOMPAQ Alphaのようなソフトウェア制御のタグ付きTLB（translation look-aside buffer）を備えたプロセッサ[7]上での実装について考える．このようなプロセッサではTLBミス時のトラップによりオペレーティングシステムに制御を移すことができるので，このときに拡張キー/ロック方式によるアクセス権のチェックを行う．

各スレッドが持つキーリストおよび各セグメントが持つロックリストは，ナノカーネルが管理する．アクセス権のチェックは，ナノカーネル内に存在するアクセス権チェックハンドラが行う．ただし，セグメントへのアクセスのたびにアクセス権チェックハンドラを呼び出すのはコストがかかること，単一アドレス空間内の任意のアドレスに対する不当なアクセスを防ぐ必要があることから，ハードウェアの仮想記憶機構を利用する．**図4**は，アドレス変換の手順を示したものである．スレッド *thr* がセグメント *seg* を呼び出す操作は，次のような手順で行われる．

(1) TLBミスにより発生するトラップで，ナノカーネル内のアクセス権チェックハンドラに制御が移る．

(2) 呼び出そうとしたセグメントのキーをスレッドがすでに継承しているか否かを調べ，すでに継承している（すでに呼出しが許可されているセグメントの別のページを参照した）場合は手順(5)へ．

(3) キーリストKL(*thr*)とロックリストLL(*seg*)を比較し，キーとロックのマッチングを調べる．呼出しが許されない場合は，例外ハンドラに処理を引き継ぐ（以下の手順は行わない）．

(4) KL(*thr*)にオブジェクトキーリストOKL(*seg*)のキーを追加する．

(5) ページテーブルの対応するエントリをTLBにセットする．

(6) スレッドに制御が移る．

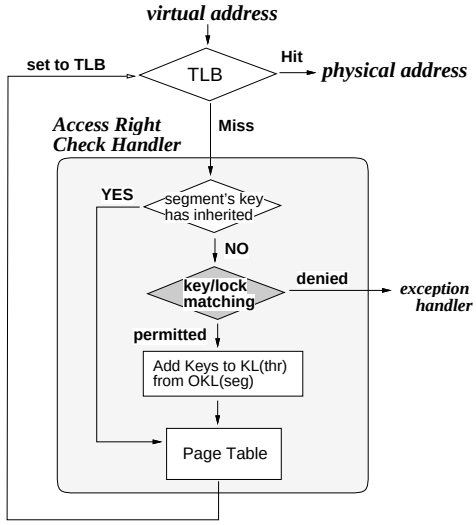

図4 アドレス変換とアクセス権チェック
Fig. 4 Address translation and access right checking.

TLBの各エントリはスレッドの識別子でタグ付けされるため，スレッド切替えの際にTLB全体をフラッシュする必要はない．セグメントに結び付けられているロックリストが変更された場合は，TLBのフラッシュが必要となるが，通常はロックリストの変更は頻繁には行われないと考えられる．

一方，スレッド *thr* がセグメント *seg* の呼出しから復帰する操作は，次のような手順で行われる．

(1) ナノカーネル内のキー没収ルーチンに制御が移る．

(2) セグメント呼出し時にオブジェクトキーリストOKL(*seg*)から継承されたキーをスレッドのキーリストKL(*thr*)から没収する．

(3) セグメント *seg* を呼び出した時点から新たにロードされたTLBエントリを無効化する．

(4) スレッドに制御が移る．

4.7 性能に関する考察

拡張キー/ロック方式は，アクセスルートコントロールという柔軟なアクセス制御を提供するが，可変長のロックが指定できるため，アクセス権をチェックする際のキーとロックのマッチングを調べるコストが大きいという欠点がある．そこで本節では，まず，拡張キー/ロック方式によるアクセス権チェックのオーバヘッドを命令ステップ数で求め，従来のアクセス制御リスト方式との比較を行う．次に，拡張キー/ロック方式の全体的なオーバヘッドの見積りを行う．

本論文で仮定している単一アドレス空間方式のオペレーティングシステムは，現在のところ実装が完了していないため，AT&TのBell研究所で開発されたオペレーティングシステムPlan9^{TM}上で動作するMIPS

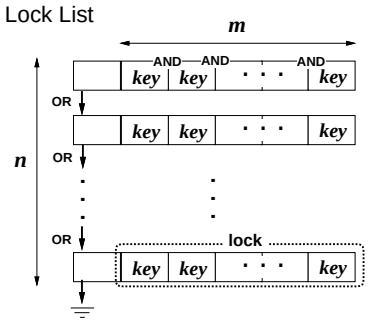

図 5 ロックリスト
Fig. 5 Data structure of lock list.

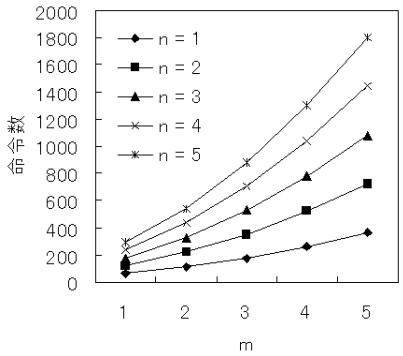

図 6 n を固定した場合の命令数の増加傾向
Fig. 6 Increase in number of instructions for fixed n.

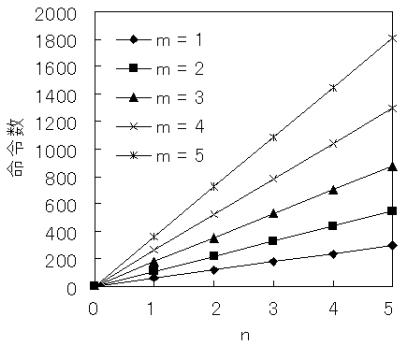

図 7 m を固定した場合の命令数の増加傾向
Fig. 7 Increase in number of instructions for fixed m.

コードシミュレータを用いてスレッドのキーリストとセグメントのロックリストのマッチングの操作に要する命令数を測定し，コストの見積りを行った．TLB ミスの処理コストや TLB ミス率については過去の研究の成果を参考にした[12),16),17)]．

4.7.1 キー/ロックのマッチングのコスト

各スレッド *thr* はキーリスト KL(*thr*) を，各セグメント *seg* はロックリスト LL(*seg*) を持つ．KL(*thr*) は配列をキースタックとして使用するように実現した．一方，LL(*seg*) は**図 5** のように，配列で実現されたロックリストエントリの線形リストとして実装した．1 つのロックリストエントリ内に指定されているキー間に対しては論理演算 AND が適用され，ロックリストエントリ間には論理演算 OR が適用される．すなわち，LL(*seg*) はキーの論理式の積和標準形でアクセス許可条件を保持する（3.3 節で述べたように論理演算を任意に用いたロックを記述できるが，ロックリストに登録される際に積和標準形に変換されることを仮定している）．

キーリスト KL(*thr*) に保持されているキーの数の合計をキーリストの長さ l とする．また，各ロックリストエントリに指定されているキーの論理式のリテラルの数の合計をロックの長さ m，ロックリストに登録されているロックリストエントリの数の合計をロックリストの長さ n とする（図 5 参照）．さらに，$m \times n$ をロックリストのサイズとする．サイズ $m \times n$ のロックリストのマッチングに要する最悪の場合のキーとロックのマッチング回数は，$\{m(m+1)/2\}n$ 回である（ただし，ここでは簡単化のために，キーリストの長さ $l = m$ とした）．ロックの長さ m を増加させた場合と，ロックリストの長さ n を増加させた場合のキーとロックのマッチングに要する命令数の増加量を調べた．**図 6** は n を固定して m を増やした場合の命令数の増加の様子を，**図 7** は m を固定して n を増やした場合の命令数の増加の様子を示している．図 6 で，m が 1 増えるごとに増加する命令数は $(17m+16)n$ 命令であり，キーとロックのマッチング 1 回あたりの命令数は 17 命令であった．一方，図 7 の直線の傾きで表される，n が 1 増えるごとに増加する命令数は，$\{17m(m+1)/2 + 16(m-1) + 41\}$ となった．また，図 7 の直線の切片で表される，アクセス権チェックを起動する際の固定のオーバヘッドは 3 命令である．したがって，サイズ $m \times n$ のロックリストが設定されているオブジェクトを呼び出す際に行われるアクセス権チェックに要する命令数は，

$$\{17m(m+1)/2 + 16(m-1) + 41\}n + 3$$
$$= \{17m^2 + 49m + 25\}n/2 + 3$$

命令となる．

次に，拡張キー/ロック方式によるアクセス権チェックのコストについて，従来のアクセス制御リスト方式との比較を試みた．3 エントリのアクセス制御リストを用いて拡張キー/ロック方式の場合と同様の実験を行った．その結果，アクセス権チェックに要する命令数は，最初のエントリでアクセスが許可される場合が

表 1 m の増加にともなう命令数の増加
Table 1 Increase in number of instructions with m.

	$m=1\to 2$	$m=2\to 3$
$n=1$	50	67
$n=2$	100	134
$n=3$	150	201

最小で 39 命令，最後のエントリでアクセスが許可される場合が最大で 81 命令であった．従来のアクセス制御リストでは拡張キー/ロック方式のようなアクセスルートコントロールを提供できないため，単純な比較はできない（両方式でコストの係数部分が異なる）が，ロックリストに登録されるロックリストエントリ数とアクセス制御リストのエントリ数を対応づけて考える．アクセス制御リスト方式では，エントリ数を n とすると，アクセス権チェックに要する最悪の場合のコストは n に比例する．したがって，m が大きくなるにつれて拡張キー/ロック方式のコストがアクセス制御リスト方式のコストに比べて増大するが，4.5 節で述べたアクセスルートコントロールの例などの考察から，実際に使用されるロックのほとんどは $m\leq 3$ であると考えられる．**表 1** は，拡張キー/ロック方式で m を大きくしたときに増加する最悪の場合（ロックリストに登録されている最後のロックリストエントリが開錠される場合）の命令数を示している．たとえば，ロックリストのサイズを 2×2 から 3×2 にした場合の命令数の増加は 134 命令である．ロックリストのサイズが大きくなるとアクセス権チェックのコストも大きくなるが，拡張キー/ロック方式によるアクセス権チェックが行われるのは，アプリケーション実行中に発生する TLB ミス時のみである．

4.7.2 セグメント呼出しと復帰のコスト

スレッドがセグメントを呼び出す（最初にアクセスする）ときのコストは，キーがすでに登録されているかどうかのチェック（4.6 節で述べた呼出しの手順 (2)）に 33 命令，キーとロックのマッチングのチェック（呼出しの手順 (3)）に $\{17m^2+49m+25\}n/2+3$ 命令（前節参照），セグメントのオブジェクトキーリストからキーを継承する操作（呼出しの手順 (4)）に $12i+41$ 命令となる．ここで，i はオブジェクトキーリストに含まれるキーの数，m はロックの長さ，n はロックリストエントリの数である．

TLB ミスに対して，TLB エントリをロードする処理（4.6 節で述べた，呼出しの手順 (1) と (5)）に要する命令数については，文献 7), 8) によると，MIPS プロセッサの場合，およそ 10〜20 命令であり，実際，MIPS プロセッサ上の Linux で計測した結果も 22 命令であった．これらの命令数はどれも，3 レベルのリニアページテーブル[16),17)]を用いた場合の，レベル 1 のページテーブルエントリ（PTE）に対するものである．文献 17) によれば，他のレベルの PTE に対するミス率は比較的小さく，その処理コストもコードのチューニングにより下げることができることから，平均のミス処理のコストは，レベル 1 の PTE に対するものとそれほど変わらない．したがって，同様のページテーブル，ミスハンドラを導入したと仮定したとき，TLB ミスの命令数をたかだか 20 命令程度と見積もることは可能であると考える．拡張キー/ロック方式を適用したときのコストについては，たとえば，オブジェクトキーリストのキーの数を 1（$i=1$），ロックの長さを 2（$m=2$），ロックリストのエントリ数を 2（$n=2$）とすると（4.5 節の例など，オペレーティングシステムの基本的なサービスでは，この程度の値がよく用いられるものと考えられる），305 命令となる．

一方，セグメントから復帰するときのコストについては，合計で平均 40 命令と見積もることができた．セグメントの呼出しと復帰のコストは，これらを合計して，約 365 命令となる．MIPS R10000 の仕様の一例として，平均の MIPS 値を 500 と仮定すると，1 秒あたりの保護ドメイン切替えの数を w としたとき，保護ドメインの切替えのオーバヘッドは $7.3\times 10^{-5}w\%$ となる．したがって，1 秒間に 10 万回のセグメント呼出し（保護ドメイン間の呼出し）を行っても[15)]，7.3%程度にオーバヘッドを抑えられることが分かる．このオーバヘッドはけして無視できるものではないが，細粒度の保護ドメインを実現し，さらにアクセスルートコントロールのような柔軟な保護機構を実現するためのコストとしては，十分実用に見合うコストであると判断される．

次に，提案方式，アクセスルートコントロールが不可能なキー/ロック方式（セグメント呼出し時のキーの継承は行うが，キーの論理式によるマッチングは行わない方式），通常のアクセス制御リスト方式のコストの比較を行う．ここでは，1×2 のロックリストと仮定する（1 つのキーで開錠できるロックだけしか設定できないので $m=1$ であり，n は上の場合と同様に $n=2$ とする）．アクセスルートコントロールが不可能なキー/ロック方式では，呼出しの手順 (3) の命令数が減じられて 145 命令になるので，合計の命令数は 254 命令となる．従来のアクセス制御リスト方式では，呼出しの手順 (2) と (4)，復帰の手順 (1)〜(4) の必要がなく，また，呼出しの手順 (3) の命令数が 81 に減じられることから，合計の命令数は 101 となる．

したがって，提案方式，アクセスルートコントロールが不可能なキー/ロック方式，アクセス制御リスト方式のコストの比は，アクセス制御リスト方式を1として，約 3.5 : 2.5 : 1 となる．オブジェクトごとにアクセス権を持たせるのに 2.5 倍，さらにアクセスルートコントロールを行うのに 1.4 倍のコストがかかることが分かる．

アプリケーション実行時の TLB ミスのオーバヘッドについては多数の報告があるが，たとえば，本章で仮定しているページテーブルの構造を用いて評価を行った文献 17) によると，2.03%から 8.88%という結果が報告されている．提案方式では，セグメントから復帰する際に TLB エントリの無効化を行っているため，通常よりも TLB ミスのオーバヘッドが増大することが予想される．ただし，オーバヘッドが増大するのは，一度呼び出したセグメントを再度呼び出す際に，本来なら TLB に登録されている可能性のあるエントリがアクセス権チェックのためにミスを起こす，という場合に限られる．しかも，TLB ミスの処理のコストは，上に述べた仮定のもとでは，セグメントの呼出し/復帰のコストのうちの 5.5%程度にすぎない．

さらに，アクセス権チェックが行われる回数を減らすための工夫も考えられる．たとえば，同じセグメントの呼出しを繰り返し行う場合，セグメント間にセキュリティ上の信頼関係があれば，グループキーを用いてセグメントを1つの保護ドメインにまとめるという方法が考えられる．また，TLB タグのビット数が十分であれば，TLB タグにスレッドの id だけではなく，スレッド id とセグメント id のペアを適用することができる．この場合，TLB エントリの無効化そのものを不要にすることができる．

5. 関連研究

アクセス権の柔軟な設定を目標とした研究が行われ，いくつかのアクセス制御機構が提案されている．その多くは，ケイパビリティ方式を拡張した手法である．条件付きケイパビリティ[4]では，ケイパビリティが使用可能な条件をケイパビリティに結び付けている．したがって，条件付きケイパビリティを用いることで，ユーザは，オブジェクトへのアクセスを特定の条件下のみに限定することができる．しかしながら，ケイパビリティが発行された時点のシステム状態がケイパビリティが有効となる条件として用いられるため，ユーザが所望の条件を直接記述したり，条件を変更したりすることができない．ケイパビリティが有効となる条件を変更するには，ケイパビリティの再作成と再配布が必要となる．さらに，拡張キー/ロック方式のように複数の条件を満たす場合に有効となるケイパビリティを作成することはできない．

Grasshopper[3]は，オブジェクト/スレッドモデルのオペレーティングシステムにおいて，柔軟なアクセス制御機構を提供している．各ローカス（スレッド）と各コンテナ（アドレス空間内の保護ドメイン）がケイパビリティを保持する．個々のケイパビリティには，アクセス権は直接記述されておらず，パーミッショングループテーブルのエントリへのポインタが格納されている．この工夫により，パーミショングループテーブルのエントリを修正すれば，ケイパビリティの再作成と再配布なしにアクセス権の無効化や変更ができる．さらに，他のコンテナへのアクセスを試みる際に，ローカスは，ローカス自身が保持しているケイパビリティと現在所属しているコンテナが保持しているケイパビリティの両者を利用することができる．この2点が拡張キー/ロック方式と類似しているが，他のコンテナに移る際に，コンテナが保持しているケイパビリティを継承できない点が拡張キー/ロック方式と異なる．Grasshopper では，ケイパビリティを継承できないため，アクセスルートコントロールは実現できない．また，論理演算子を用いて複合的なアクセス許可条件を設定するような機能も提供していない．

Angel[9]は，単一アドレス空間オペレーティングシステムにおいて，柔軟なアクセス制御機構を提供している．Grasshopper と同様に，biscuit と呼ばれるケイパビリティにはアクセス権は直接記述されておらず，オブジェクトへのアクセス条件が記述された ACD（Access Control Descriptor）へのポインタが格納されている．アクセス条件として，あらかじめ保護ドメインに取り込まれていなければならないオブジェクトを論理演算子 AND と OR を用いて指定することができる．この点が拡張キー/ロック方式と類似している．しかしながら Angel では，拡張キー/ロック方式が提供している，サブジェクトが動的にキーを獲得する機能や呼び出したオブジェクトからキーを継承していく機能は提供されていない．

もう1つ別のアプローチとして，アクセス制御リスト方式を拡張する手法がある．CACL[11]（Capabilities and Access Control Lists）は，オブジェクト指向データベースシステムにおいて，細粒度で効率的な保護機構を提供している．CACL のアクセス制御モデルは，アクセス制御リスト方式に基づいている．アクセス制御機構を型システムと統合し，コンパイル時にアクセス権のチェックをすることで，メソッド単位の粒度の

細かいアクセス権の設定を効率的に提供している．しかしながら，拡張キー/ロック方式で提供するアクセスルートコントロールのような柔軟なアクセス制御は実現されていない．

Cubix[10]は，単一アドレス空間内に配置されたセグメントの保護を効率的に行うために，特殊なMMU（memory management unit）を用いた実装を行っているが，アクセス制御の基本的な手法はアクセス制御リスト方式である．

6. おわりに

互いに協調して働く複数のオブジェクトによって構成される，拡張可能なオペレーティングシステムでは，柔軟で細粒度なアクセス制御機構が要求される．我々は，アクセス制御にオブジェクト間の協調関係とサブジェクトのアクセス履歴を採り入れた拡張キー/ロック方式を開発した．拡張キー/ロック方式により，従来のアクセス制御方式では不可能であったアクセスルートコントロールという柔軟なアクセス制御が実現された．

次に，拡張キー/ロック方式を拡張可能なオペレーティングシステムに適用することについて検討を行った．拡張キー/ロック方式は柔軟なアクセス権の設定が可能である反面，従来の方式よりもアクセス権チェックに要するコストが大きい．しかしながら，ソフトウェア制御のタグ付きTLBを備えたプロセッサ上で実装を行うことで，アクセス権チェックが必要となる機会を削減し，効率化を図ることができる．命令数をもとにした性能の見積りにより，拡張キー/ロック方式によるアクセスルートコントロールが実用的なコストで実現できることを示した．

本論文では，単一仮想アドレス空間のマップ情報を単一のページテーブルで管理する方式を仮定した．その対案として，スレッドごとにページテーブルを持たせ，スレッドのアクセス権をページテーブルに反映させる方式も考えられるが，その方式の性能評価および比較は今後の課題である．

また，本論文では，拡張キー/ロック方式をオペレーティングシステムに適用したが，複数のオブジェクト間の協調関係とサブジェクトのアクセス履歴に基づいたアクセスルートコントールは，モバイル計算環境や分散資源管理にも適用できると考えられる．

参考文献

1) Bershad, B., Savage, S., Pardyak, P., Sirer, E.G., Becker, D., Fiuczynski, M., Chambers, C. and Eggers, S.: Extensibility, Safety and Performance in the SPIN Operating System, *Proc. 15th ACM Symposium on Operating System Principles*, pp.267–284 (1995).
2) Chase, J.S., Levy, H.M., Feeley, M.J. and Lazowska, E.D.: Sharing and Protection in a Single Address Space Operating System, *ACM Trans. Computer Systems*, Vol.12, No.4, pp.271–307 (1994).
3) Dearle, A., di Bona, R., Farrow, J., Henskens, F., Hulse, D., Lindstrom, A., Norris, S., Rosenberg, J. and Vaughan, F.: Protection in Grasshopper: A Persistent Operating System, *Proc. 6th Int. Workshop on Persistent Object Systems*, pp.60–78 (1994).
4) Ekanadham, K. and Bernstein, A.J.: Conditional Capabilities, *IEEE Trans. Softw. Eng.*, Vol.SE-5, No.5, pp.458–464 (1979).
5) Grimm, R. and Bershad, B.N.: Access Control in Extensible Systems, techinical report, Dept. of Computer Science and Engineering, University of Washington, UW-CSE-97-11-01 (1997).
6) Hamilton, G. and Kougiouris, P.: The Spring Nucleus, *Proc. Summer USENIX Technical Conference*, pp.147–159 (1993).
7) Jacob, B.L. and Mudge, T.N.: Software-Managed Address Translation, *Proc. 3rd Int. Symposium of High Performance Computer Architecture*, pp.156–167 (1997).
8) Jacob, B.L. and Mudge, T.N.: A Look at Several Memory Management Units, TLB-Refill Mechanism, and Page Table Organizations, *Proc. 8th Int. Conference on Architectural Support for Programming Languages and Operating Systems*, pp.295–306 (1998).
9) Murray, K., Wilkinson, T., Osmon, P., Saulsbury, A., Stiemerling, T. and Kelly, P.: Design and Implementation of an Object-Oriented 64-bit Single Address Space Microkernel, *Proc. USENIX Symposium on Microkernels and Oteher Kernel Architectures*, pp.31–43 (1993).
10) Okamoto, T., Segawa, H., Shin, S.H., Nozue, H., Maeda, K. and Saito, M.: A Micro Kernel Architecture for Next Generation Processors, *Proc. USENIX Workshop on Micro-kernels and Other Kernel Architectures*, pp.83–94 (1992).
11) Richardson, J., Schwarz, P. and Cabrera, L.: CACL: Efficient Fine-Grained Protection for Objects, *Proc. OOPSLA '92*, pp.263–275 (1992).
12) Rosenblum, M., Bugnion, E., Herrod, S.A., Witchel, E. and Gupta, A.: The Impact of Architectural Trends on Operating System Performance, *Proc. 15th ACM Symposium on Op-*

erating System Principles, pp.285–298 (1995).
13) Shigeta, S., Tanimori, T., Shimizu, K. and Ashihara, H.: A Fine-Grained Protection Mechanism in Object-Based Operating Systems, *Proc. Int. Workshop on Object Orientation in Operating Systems*, pp.156–160 (1996).
14) Shigeta, S., Okamoto, S., Shimizu, K. and Sowa, M.: A Flexible Access Control Mechanism Based on the Key/Lock Scheme, *Proc. Int. Technical Conf. in Circuits/Systems, Computers and Communications*, pp.1385–1388 (1999).
15) 品川高廣，河野健二，高橋雅彦，益田隆司：拡張コンポーネントのためのカーネルによる細粒度軽量保護ドメインの実現，情報処理学会論文誌，Vol.40, No.6, pp.2596–2606 (1999).
16) Tallui, M., Hill, M.D. and Khalidi, Y.A.: A New Page Table for 64-bit Address Space, *Proc. 15th ACM Symposium on Operating System Principles*, pp.184–200 (1995).
17) Uhlig, R., Nagel, D., Stanley, T., Mudge, T., Sechrest, S. and Brown, R.: Design Tradeoffs for Software-Managed TLBs, *ACM Trans. Computer Systems*, Vol.12, No.3, pp.175–205 (1994).

(平成 12 年 12 月 19 日受付)
(平成 13 年 4 月 6 日採録)

繁田 聡一（正会員）

1972 年生．1995 年電気通信大学電気通信学部情報工学科卒業．1997 年同大学大学院電気通信学研究科情報工学専攻博士前期課程修了．2000 年同大学院情報システム学研究科情報ネットワーク学専攻博士後期課程単位取得退学．現在，同研究科教務職員．システムソフトウェア，コンピュータシステムのセキュリティの研究に興味を持つ．電子情報通信学会員．

清水謙多郎（正会員）

1957 年生．1985 年東京大学大学院理学系研究科情報科学専攻博士課程修了．理学博士．1998 年より東京大学大学院農学生命科学研究科教授．オペレーティングシステム，並列・分散処理，生命情報科学の研究に従事．ACM，IEEE Computer Society，電子情報通信学会，日本ソフトウェア科学会，日本シミュレーション学会，生物物理学会，日本バイオインフォマティクス学会，日本農芸化学会各会員．

曽和 将容（正会員）

1943 年生．1974 年名古屋大学大学院博士課程修了，工学博士．同年群馬大学工学部助手．助教授を経て 1986 年名古屋工業大学工学部教授．1993 年から電気通信大学大学院情報システム学研究科教授．この間，並列コンピュータ，データフローコンピュータ，並列コントロールフローコンピュータ，並列プロセッサ，コンピュータネットワーク等新しいコンピュータシステムの研究に従事．最近は，並列分散ユビキタスコンピューティングの研究に興味を持っている．電子情報通信学会，日本ソフトウエア科学会，IEEE，ACM 各会員．