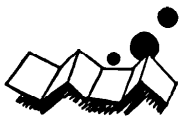
解 説



オブジェクト指向とオペレーティングシステム†

土 居 範 久††

1. はじめに

オペレーティングシステムの分野では古くからセグメントあるいはケーパビリティといった現在のオブジェクト指向の考えに近い概念が取り入れられている．そこで本稿ではこれらの概念および実現方法について具体例をとおして解説する．

まず，**2.** で'もの'を単位としたプログラミングを可能にしたセグメンテーションおよびそれを支援する'記述子によるアドレシング'について述べ，**3.** で'もの'の抽象化と管理を容易にするケーパビリティの概念およびそれを支援する'ケーパビリティによるアドレシング'について述べる．そして，**4.** で'もの'を単位として取り上げるようになった重要な背景の一つである保護特にアクセス制御について述べてから，最後に，**5.** でハードウェアアーキテクチャの基本はケーパビリティによるが'抽象オブジェクト'を基本単位としてオブジェクト指向的システムを提供する'オブジェクトによるシステム'について解説する．

2. 論理アドレス空間の切断

まず，'もの'を単位としたプログラミングを可能にしたものとして**セグメンテーション** (segmentation) がある．これは，記憶を語 (word) の線形配列として捉えずに可変長の**セグメント** (segment) の集合として捉えようとするものである．セグメントとは，手続きとか配列といった論理的な単位としての'もの'を表現する連続した記憶である．ページングが物理アドレス空間 (physical address space) を切断し，オペレーティングシステムでの物理アドレス空間の管理を容易にする技法であるのに対し，セグメンテーションは，論理アドレス空間 (logical address space) を切断することによって，記憶を，独立にアドレス可能なセグメントの集合として提供しようとするものである．プログラムでは，いちいちのアドレスは，個々のセグメントに固有の**セグメント番号** (segment number) とそのセグメントの先頭からの**相対位置** (offset) で指定する．つまり，このセグメント番号と相対位置の対が**論理アドレス** (logical address) になる．オペレーティングシステムでは，プログラムの実行に必要とするセグメントだけを主記憶にロードしさえすればよく，しかも，その記憶位置はその都度もっとも都合のよい場所にする (再配置 (relocation) する) ことができる．このことによって，仮想記憶 (vertual memory) が実現可能になる．

プログラムでは，物理アドレスを用いずに論理アドレスを用いることで，再配置を容易に可能にしているのだが，実行に際しては，この論理アドレスを物理アドレスに変換する必要がある．この変換を**アドレス変換** (address mapping) といい，そのために，論理アドレスと物理アドレスとを対応づける**セグメント表** (segment table) と呼ぶ変換表 (mapping table) を用いる．セグメント表のいちいちの欄を**セグメント記述子** (segment descriptor) といい，個々のセグメントに対し一つずつ用意する．セグメント記述子は，主記憶内での所在や各種制御情報を含む．セグメントを再配置するときには，オペレーティングシステムではこのセグメント記述子を修正するだけでよい．このようなアドレシング方式を，一般に，**記述子によるアドレシング** (descriptor-based addressing) という．

セグメンテーションの利点をまとめてみると，次のとおりである[1]．

(1) セグメントはプログラミング上の論理的な'もの'と対応している．

(2) セグメンテーションは実行時に論理的な'もの'の伸縮を可能にする．

(3) セグメンテーションは主記憶の再配置および

† Object-oriented approaches to operating systems by Norihisa DOI (Institute of Information Science, Keio University).

†† 慶應義塾大学情報科学研究所

仮想記憶を支援する.

(4) セグメントはプログラム間あるいはプロセス間での論理的な'もの'単位の分割, 保護, 共用を可能にする.

記述子によるアドレシング方式を採用した代表的なものに, バロース社の B 5000, J. K. Iliffe の Rice University Computer および Basic Language Machine, MIT の Multics がある.

2.1 B5000[1),2)]

記述子によるアドレシングを初めて採用した機械で, 1961 年に開発された. Algol 60 を基本言語としており, その後のバロース社の機械のもとになった.

プログラムはコードセグメントとデータセグメントとで構成される. これらのセグメントは一つずつとは限らない. そして, プログラムの実行時のアドレシング環境は, セグメント, 固有のスタックおよび固有の**プログラム参照表** (program reference table) で定義される. プログラム参照表には, セグメント記述子とスカラーデータが含まれる. このプログラム参照表が, 各プログラムの実行時の**ドメイン** (domain) を定義することになる. プログラム参照表を用いてプロセスのアドレス空間を分離するアドレシング機構が, その後のケーパビリティの概念に影響を与えている.

2.2 Rice University Computer[1),3)]

1959 年に開発が開始された Rice 大学のこの機械は, **コードワード** (codeword) に基づいたセグメント機構を採用している. コードワードは論理的な単位に対する記述子である. おのおののセグメントは命令コードだけか命令コードとデータかデータだけかコードワードだけかで構成される. データの場合, 1 次元の配列を一つのコードワードで指すような機構になっているので n 次元の配列には n 段階のコードワードを用いることになる. 配列の始点となるコードワードはスカラーデータと一緒に一つの表としてもつ. そして, これらの表や手続き, 大域的な名前をもつ記号表などは**主コードワード** (principal codewords) と呼ぶあらかじめ定められた位置にあるコードワード表に納めたコードワードで指すという多段階の間接指定方式をとっている. データを引数として他の手続きに送るときには対応するコードワードを指定する. したがって, コードワードは, 'もの' つまりオブジェクトの名前として用いられていると考えることもできる. コードワードは, ケーパビリティの概念にきわめて近い.

Basic Language Machine[4)] は, この Rice University Computer を発展させたもので, やはりコードワードによるセグメンテーション方式を採用している.

2.3 Multics[5),6)]

B 5000 方式のセグメンテーションの場合には, セグメントを共用させようとしたとき, かなりの制約が付く. つまり, プロセス間で共用のセグメントや, 個々のプロセスに固有のセグメントであってもそれが共用のセグメントから参照されているセグメント, などのセグメント番号はあらゆるプロセスで同じである必要がある (**図-1** 参照). つまり, これらのセグメントは, これらを共用する個々の論理空間内で同じ位置に置かれなければならないのである. これでは, 融通性に欠けることから, プロセスごとの論理空間内の任意の位置に共用のセグメントといえども置けるようにしたセグメンテーション方式をとるようにしたのが Multics である. Multics での詳細はさておいて, 概念的には次のようになる (**図-2** 参照)[6)].

おのおののプロセスに対して, 個々の手続きごとに一つずつリンケージセグメントが用意される. そして, このリンケージセグメントにセグメント表のインデクスがあり, 共用セグメントのセグメント番号はリンケージセグメント内の相対位置として扱われる. そ

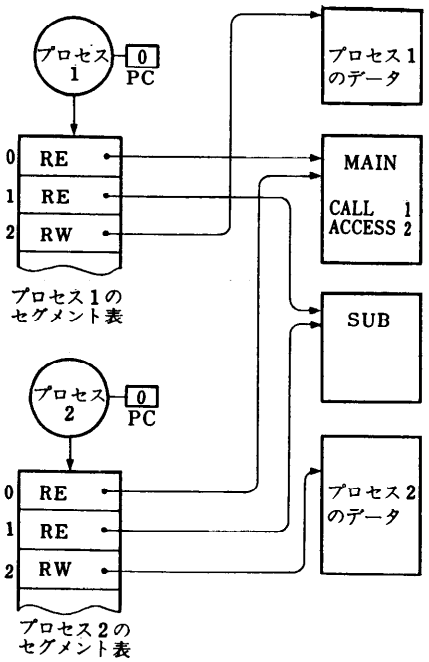


図-1 B5000 のセグメンテーション (概念図)[6)]

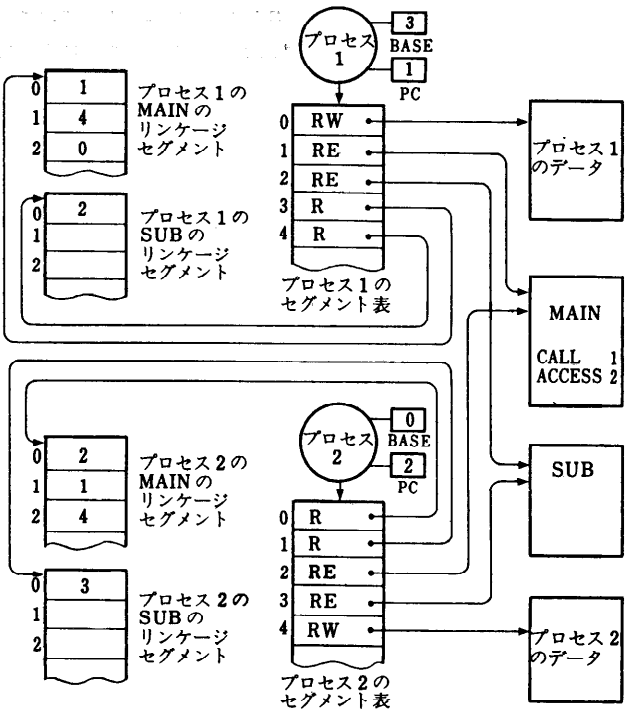


図-2 Multics のセグメンテーション（概念図）[6]

こで，プロセス1で'CALL 1'に相当する命令を実行したときには，リンケージセグメントベースレジスタ BASE が指している セグメント表内の セグメント記述子が指すリンケージセグメントの第1語が取り出される．この値4がベースレジスタに設定される．このベースレジスタの指すリンケージセグメントの第0語にある2が PC に設定され制御が共用手続き SUB に移る．しかし，手続き間で引数として渡されるセグメントアドレスは，個々の手続きに従属するリンケージセグメントのインデクスではなく一つのプロセスの論理アドレス空間を定義するセグメント表のインデクスでなければならないので，セグメントアドレスはコンテキストに依存することになる．

3. ケーパビリティ

一般に，以上のシステムでは，論理アドレスは一つのプロセス内でしか有効でない．さらに，長期にわたって保存しておきたい'もの'は，ファイルシステムを用意し，名前の付け方を始めとしてセグメントとは全く違った体系をとっていた．プロセスが存在している間しか存在しない短期の'もの'も，長期にわたって保存しておく'もの'も，名前の付け方や保護の仕方を統一することによって，コンテキストに依存することなく取り扱えるようにしようとするものが**ケーパビリティ** (capability, 資格) である[1),7)]．ケーパビリティは，システム内のあらゆる'もの'つまり**オブジェクト** (object) をアクセスする際にいちいち提示し，そこに記載されている権利の範囲内で，そのオブジェクトをアクセスさせてもらうための，いわば'切符'のようなものである．ケーパビリティは一意に決まる**オブジェクト名** (object identifier) と**アクセス権** (access right) とからなる．アクセス権とは，そのオブジェクトに対して許されている操作である．そして，おのおののユーザあるいはプロセスには，アクセスすることが許されているすべてのオブジェクトに対するケーパビリティのリストすなわち**ケーパビリティリスト** (capability list) を関連づける．このケーパビリティリストがドメインを定義する．オブジェクトは，このリスト内のインデクスを用いて指定する．オペレーティングシステムでは，ユーザがケーパビリティリストを直接書き換えることができないようにすることによってシステムの完全性を保つ．ケーパビリティを用いたシステムを**ケーパビリティによるシステム** (capability-based system) といい，ケーパビリティ

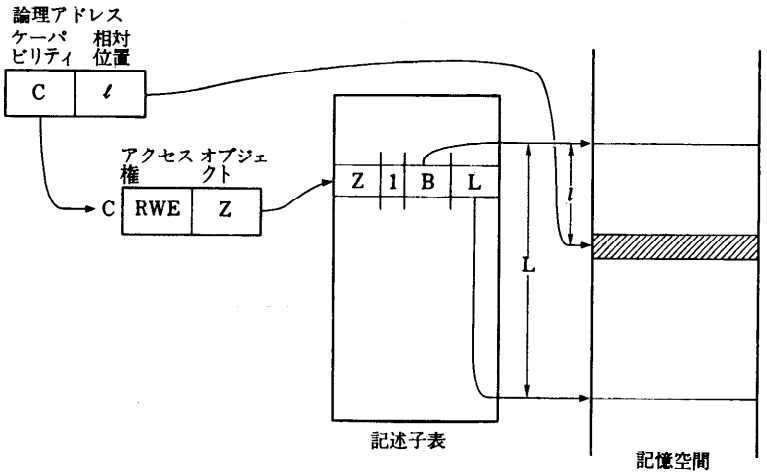


図-3 ケーパビリティによるアドレシング

を用いた記憶のアドレシング方式を**ケーパビリティによるアドレシング** (capability-based addressing) という（**図-3** 参照）．オブジェクトに一意な名前を付けることによりオブジェクトを単一レベルでアドレシングすることが可能になる（二次記憶の概念がなくなる）．

個々の手続きごとにケーパビリティリストを用意すれば，その手続きの実行時のドメインを限定できるので，その手続きを呼び出したプロセスからその手続きに局所的なオブジェクトを保護することができるだけでなく，呼び出された手続きが呼び出した側に局所的なオブジェクトをみだりにアクセスすることから保護することもできる．このような手続きを**保護付き手続き** (protected procedure) とか**保護付きサブシステム** (protected subsystem) という．このドメインを切り替えるためのケーパビリティとして enter がある[7]．enter ケーパビリティはケーパビリティリストを指し，そこで定義されている手続きの一つを呼び出すことを許す．呼び出したときには，そのケーパビリティリストで定義されたドメインに移行する．手続きの実行が終了すると，もとのドメインに戻る．

あらゆる資源のハンドラを，この保護付き手続きを用いて作り，オペレーティングシステムを保護付き手続きの集合として構成することができる．このようにしたときには，いわゆる特権モードの必要がない（すべての特権は，ケーパビリティをもっているかどうか，そのアクセス権はなにか，で決まる）．さらに，資源のタイプによっては，その型板を用意しておき，オペレーティングシステムを呼び出すことによって，その資源を作りだすことができるようにしておけば，きわめて都合がよい．このような機能があれば，ユーザはオペレーティングシステムと統一の取れた方法でオペレーティングシステムの機能を拡張することもできる．

こうなると，'オブジェクト' はオブジェクト指向プログラミング言語でいう 'オブジェクト' と概念的に変わるところがない．プログラミング言語での 'オブジェクト' の保護付き版にほかならない．

ケーパビリティの概念は1966年に J. B. Dennis と E. C. Van Horn によって提案された[7]．当初は，主記憶のアドレシングにしか用いられなかったが，しだいに広く用いられるようになった．ケーパビリティを用いた初期のシステムとしては，MIT の PDP-1 TSS[8]，Chicago 大学の Chicago Magic Number Machine，California 大学 Berkeley の CAL-TSS などがある．Chicago Magic Number Machine がケーパビリティを支援するために設計された最初の機械である．さらに，ケーパビリティの概念を整然と整理したので有名なものとして英国 Plessey 社の Plessey System 250，Cambridge 大学の CAP[9] がある．

3.1 CAL-TSS[1),10)]

CAL-TSS は 1968 年に California 大学 Berkeley で CDC 6400 を使って汎用のケーパビリティによるオペレーティングシステムを実現する目的をもって開発が開始されたシステムである．汎用の機械を使っていることからケーパビリティを支援する特別なハードウェアアーキテクチャはない．CAL-TSS のケーパビリティは，オブジェクトのタイプ，遂行できる操作，オブジェクトの一意名とその所在から構成されている．システムには次のようなタイプのオブジェクトが用意されており，オペレーティングシステムを呼び出すことで，これらのタイプのオブジェクトを作り操作することができる．

（1） ファイル

（2） プロセス間通信用のイベントチャネル

（3） システム資源を使うための権限を得るためのアロケーションブロック

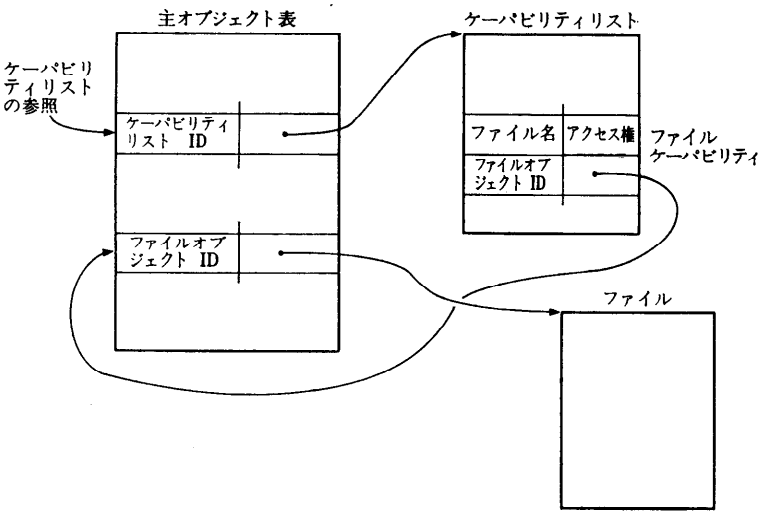


図-4 CAL-TSS のアドレシング機構[1)]

(4) ケーパビリティリスト
(5) プロセス間で共用するドメインを同定するラベル
(6) 保護付き手続き(オペレーションという)
(7) プロセス
(8) 1語のデータを封印するのに用いるタイプ

CAL-TSS の特徴として次の二つをあげることができる．一つはケーパビリティとアドレシング情報との分離である．**図-4** に示すような2段構えのアドレシング機構をとっており，オブジェクトの主記憶内での再配置およびオブジェクトの削除を容易にしている．もう一つは，オブジェクトの一意名は，たとえ削除された後でも再使用せず，全く違う別のオブジェクトを指すことがないようにしていることである．

3.2 Plessey System 250[1),11)]

Plessey 250 は，そもそも高信頼性の電話回線交換システムとして設計された．オペレーティングシステムの設計の基本概念は層構造とデータ抽象にある．主記憶はセグメント化されており，一つのセグメントにはケーパビリティかデータかのいずれかしかもつことができない．また，レジスタもデータ用とケーパビリティ用の2種類がある．(これらの点に関しては，Chicago Magic Number Machine も同じである[1)]．) セグメント内のデータをアクセスするには，プログラムで，まず，そのセグメントに対するケーパビリティをケーパビリティレジスタの一つにロードしておかなければならない．ケーパビリティセグメントはネットワーク状に構成でき，これらがドメインを定義する．ケーパビリティレジスタ内のケーパビリティは，主記憶内のそのセグメントの位置，セグメントのサイズ，アクセス権からなる．システム内のあらゆるオブジェクトに対して一つずつケーパビリティがあり，これらは一つの表(**システムケーパビリティ表** (system capability table, SCT と略記) と呼ばれる) にまとめられている．プロセスごとのケーパビリティはケーパビリティセグメントにまとめられており，そのいちいちのケーパビリティはアクセス権とシステムケーパビリティ表内での所在とからなる．SCT のサイズを制限し記憶の使用効率化を計るために，長期間使用されないセグメントに対する SCT 記入欄は二次記憶に追い出すことをしている．そのために，ガベージコレクタを随時走らせ SCT を管理している．システムのあらゆる資源は，その状態をもつデータセグメントとその資源を操作する一群の手続きとからなる保護付きサブシステムで実現されている．

4. 'もの' の保護

オペレーティングシステムで'もの'を単位として取り上げるようになった重要な背景の一つに保護がある．特にオペレーティングシステムではシステム内の'もの'への直接のアクセスは，すべて正当と認められたものであることを保証する必要がある．これを保証することを，一般に，**アクセス制御** (access control) という．

4.1 アクセス行列モデル[12),13)]

システムのアクセス制御状態Sは，おのおのの**オブジェクト** (対象) への個々の動作の**主体** (subject) のアクセス許可を表現したものであり，アクセス制御を実施する機構では次の二つを行わなければならない．

(1) Sで許可されているアクセスだけを各主体に許す
(2) Sの変更は許可されているものだけを許す

この概念は**アクセス行列モデル** (access matrix model) で表現できる．アクセス行列モデルは，**状態** (state, 保護状態 (protection state)) および**状態遷移** (state transition) で定義する．

〔a〕 保 護 状 態

保護状態は次の三つで表現する．

(1) 主体 (またはドメイン) ユーザ，プロセスあるいは手続きといったアクセスを行うもの (またはプロセスが実行される保護環境であるドメイン)．
(2) オブジェクト それへのアクセスが制御され

		オブジェクト								
		主体			ファイル			端末		
		s_1	s_2	s_3	f_1	f_2	f_3	t_1	t_2	
主体	s_1		control		owner read write	*read	execute	send receive		
	s_2			wakeup block		+read write			send receive	
	s_3		wakeup block				read write execute			

図-5 アクセス行列

なければならないもの．主体（またはドメイン）も保護されるべきオブジェクトである．各オブジェクトはシステムで定義された一意名をもつ．

（3） アクセス行列A　行が主体（またはドメイン）に対応し列がオブジェクトに対応する．$A[s, x]$には，主体s（またはドメインsのプロセス）に対して許されているオブジェクトxへのアクセスの種類すなわちアクセス権の並びがある．アクセス行列の例を示すと**図-5**のようになる．

オブジェクトの種類ごとに**モニタ** (monitor, オブジェクトモニタ (object monitor)) を設け，そのオブジェクトへのアクセスを制御する．オブジェクトxに対するモニタでは，sのxへのアクセス要求があったとき，その要求が$A[s, x]$にあるか否かを確かめ，許されていない要求の場合には拒否する．

〔b〕 アクセス権

アクセス権は，オブジェクトに対して遂行できるアクセスの種類を規定するもので，オブジェクトの種類によって異なる．たとえば，セグメントやファイルに対するアクセス権としては読み取り (read)，書き出し (write)，実行 (execute) がもっとも一般的である．

〔c〕 状態遷移

保護システムの状態遷移はコマンドで記述する．たとえば，M. A. Harrison らは基本命令の列でコマンドを指定する方法をとっている[14]．これらの基本命令は保護状態を管理するモニタによって制御される．

4.2 アクセス制御機構[12),13)]

アクセス制御は，システムのいちいちの状態を，確実に，抽象モデルの許可された状態に対応させることによって行う．しかし，保護システムを実現する際には，アクセス行列をそのまま実現することはまずない．アクセス行列は非常に大きくしかも実際にアクセス権が指定されている欄は疎にしかないからである．実現されているアクセス制御機構の基礎概念として，**アクセス階層** (access hierarchy)，**アクセス制御リスト** (access control list)，**ケーパビリティ**の三つがある．

〔a〕 アクセス階層

代表的機構として，**特権モード** (privileged mode) と**保護リング** (protection ring) がある．

〔b〕 アクセス制御リスト

アクセス制御リストは，個々のオブジェクトxへのアクセスが許されている主体と許されているアクセス権の対を並べたもので，並びのi番目には主体s_iと$A[s_i, x]$の対がある．したがって，アクセス行列の列xの空でない欄を表現したいものになる．許可リスト (authorization list) とかアクセスリスト (access list) ともいう．また，アクセス制御リスト方式を**リスト主導** (list-oriented) 方式ということもある．

アクセス制御リストはファイルシステムによく用いられる．ファイルを代表とするいわゆる所有物については，所有者はそのオブジェクトに対する他のユーザの記入欄を削除できるので，アクセス制御リストは所有物に対するアクセスを完全に制御できるという特徴をもつ．ただし，他のユーザが現在使用中のオブジェクトについては，その使用者のアクセス権を取り消しても，即座にその効果が現れるとは限らない．

〔c〕 ケーパビリティ

ケーパビリティは，オブジェクトxの一意名とそれに対するアクセス権との対で，その並びがケーパビリティリストである．したがって，ケーパビリティリストは，アクセス行列の行の空でない欄を表現したものになる．ケーパビリティリストがドメインを定義する．ケーパビリティ方式は**チケット主導** (ticket-oriented) 方式ともいう．

（1） 増幅　切り替える前のドメインではもっていなかったアクセス権を，切り替える先でのドメインではもたせることを，ケーパビリティの**増幅** (amplifica-

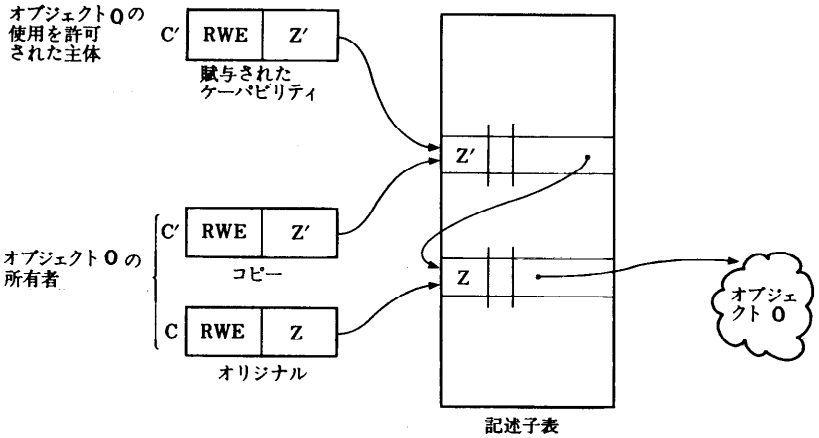


図-6 ケーパビリティの間接指定

tion) という.

(2) 取消し ケーパビリティを賦与することによって,オブジェクトを多数の主体で共用することが容易にできる.しかし,そのオブジェクトがだれかの所有物で,その所有者が他の人のアクセス権を取り消す(revoke)ことができるようにしようとすると面倒なことが生じる.賦与してしまったケーパビリティを探さなければならないからである.このための解決策として,間接指定による方法が提案されている(**図-6** 参照).

5. オブジェクトによるシステム

これまでに述べてきたシステムは,データ抽象化およびオペレーティングシステムの系統的拡張を支援するものではあったが,これらの考えをより進め '抽象オブジェクト' を基本単位としオブジェクト指向的なシステムを提供するシステムが 1970 年代に入ると登場してきた.さらに 1980 年代に入るとこれらのアイデアをハードウェアアーキテクチャで支援する商用のシステムすら登場してきた.これらのシステムを**オブジェクトによるシステム** (object-based system) と呼ぶことがある.Hydra, StarOS といった Carnegie-Mellon 大学で開発されたオペレーティングシステムが前者の代表であり,IBM 社の System/38, Intel 社の iAPX 432 が後者の代表である.いずれにせよ抽象オブジェクトはセグメントで構成されることから,セグメントがケーパビリティでアドレスされる基本単位である.

5.1 Hydra[15),16)]

Hydra は 16 台の PDP-11 をクロスバスイッチでつないだマルチプロセッサシステム C.mmp のオペレーティングシステム核である.システム内のあらゆるものはオブジェクトであり,オブジェクトはアドレシングおよび保護の基本単位であって,ケーパビリティで一意づけられる.新しいオブジェクトタイプ,それらのタイプの**個体** (instance) およびケーパビリティを作ったり操作したりすることができる環境の提供を目的としている.オブジェクトは,オブジェクト名,このオブジェクトに対する操作を定義するタイプ,オブジェクトの状態を表す表現 (representation, データ部とケーパビリティリスト) から構成される.タイプは資源を表し,その資源のすべての個体に対する型板 (**タイプオブジェクト**と呼ぶ) は新しい個体を作り出すのに必要とする情報と個体を操作する手続きに対するケーパビリティからなる.新しい個体を作りたければ,そのタイプオブジェクトに $CREATE 呼出しをかけると,タイプオブジェクトが個体を作りそれに対するケーパビリティを返す.システムにあらかじめ用意されているタイプは次のとおりである.

(1) プロセス (2) 手続き (3) 局所名域
(4) ページ (5) セマフォ (6) ポート
(7) 入出力装置 (8) 方策 (policy)
(9) データ (10) ユニバーサル (universal)
(11) タイプ

タイプオブジェクトも含め,すべてのオブジェクトはタイプオブジェクトで表現されている.つまり Hydra では,すべての資源はオブジェクトとして表現され,オブジェクトはドメイン間で受け渡すことができる.ユーザはタイプオブジェクトをもとに新しく資源を作り出し,作り出したこれらの資源の個体はタイプ別に用意した手続きで制御できる.

手続きはすべて保護付き手続きであり,自分自身の実行時のドメインをもっている.手続きとその実行とを区別するために手続きの静的な表現である手続きオブジェクトと実行時の活動記録 (activation record) である**局所名域** (local name space, LNS と略記) オブジェクトがある.実行時のドメインを定義するケーパビリティリストは手続きオブジェクトにある.その手続きで直接アドレス可能なオブジェクトはすべて LNS のケーパビリティリスト内にケーパビリティがある.このケーパビリティは呼び出された手続きから継承したものと引数として渡されたもので初期設定される.

個体に操作を加えたければ,その元であるタイプオブジェクトにその個体のケーパビリティを送り,加えたい操作を指定する.あらゆるタイプのオブジェクトに共通な操作があるが,これらはシステムに用意されている (汎用操作 (generic operation) という).

個々の個体に対するケーパビリティでは,その所有者がオブジェクトの表現を直接アクセスすることは禁止しなければならないが,タイプオブジェクトで操作を加えるときにはそれが可能でなければならない.つまり機能を増幅する必要がある.そのために**増幅型板** (amplification template) と呼ぶ一種のケーパビリティを用いている[17].増幅型板にはタイプおよび '必要アクセス権' と '増幅アクセス権' とが含まれており,手続き呼出しが行われると引数として渡されたケーパビリティと増幅型板とを比べ,タイプが同じでしかも

'必要アクセス権' を満足しているときにかぎり '増幅アクセス権' をアクセス権とするケーパビリティが使われることになる.

また, Hydra ではオブジェクトの名前は常に一意であり, あらゆるオブジェクトは単一レベルでアドレシングされる.

Intel 社の iAPX 432 はプロセッサのスケジューリング, プロセス間通信, 記憶割り当てといったオペレーティングシステムの機能をハードウェアおよびマイクロコードで実現するだけでなく Hydra の概念をハードウェア的に支援しようとしたシステムであるということができる[18].

5.2 IBM System/38[1),19),20)]

1980 年に出荷が開始された IBM 社の System/38 は COBOL, RPG III あるいはデータベースといった商用のシステムを実現したオブジェクトによるシステムである. System/38 では 14 種のシステムオブジェクトを提供している. システムオブジェクトは, オブジェクトの表現を保持する機能部 (functional portion) とユーザプログラムで直接操作が可能なデータを保持するスペース部 (space portion) からなる. これらの各部はそれぞれセグメントから構成されている. タグ方式をとっているのでケーパビリティ (ポインタ (pointer) と呼ぶ) はスカラーデータと一緒にセグメントに格納できる.

ケーパビリティシステムでは一度与えた権限を取り消すことが難しいことはすでに述べたが, System/38 では, これを解決するためにアクセス制御リストを使い 2 種類のケーパビリティを用いることで部分的に解決している. つまり, アクセス権を含まないケーパビリティ (非認可 (unauthorized) ケーパビリティ) とアクセス権を含むケーパビリティ (認可 (authorized) ケーパビリティ) の 2 種類のケーパビリティがあり (どちらにするかはオブジェクトの所有者が決める), 非認可ケーパビリティの場合には, そのオブジェクトに付随するアクセス制御リストがチェックされるのである.

System/38 でも保護付き手続きのための機構は用意されているが, これまでのものと多少異なる. プロセスが手続きを呼び出すと, その手続きは呼び出した手続きのドメインで走る. ただし, 手続きを作るときにその所有者は自分のドメインを呼び出したプロセスのドメインに加えるよう指定できるので (プロファイルの借入 (profile adoption) という), これを使えばケーパビリティの増幅ができる.

6. おわりに

以上, 記述子によるアドレシング, ケーパビリティによるアドレシング, オブジェクトによるシステムおよびアクセス制御について述べた. ケーパビリティにより 'もの' を単位として扱うことができることで, プログラミングの観点からもオペレーティングシステム設計の観点からも数多くの利点がもたらされる半面, 解決されなければならない問題も出てきている. これらの問題としては次のようなものがある[1),10)].

(1) かなりの記憶域を必要とする.
(2) オブジェクトのための空間およびオブジェクト名に関してガベージコレクションを必要とする.
(3) 保護付き手続き機構が高価につく.
(4) 1 回のアクセスあたりの主記憶参照回数がかなりになる.

参考文献

1) Levy, H. M.: Capability-Based Computer Systems. Digital Press (1984).
2) Organick, E. I.: Computer System Organization. Academic Press (1973). (土居範久訳, 計算機システムの構造. 共立出版 (1978)).
3) Iliffe, J. K. and Jodeit, J. G.: A Dynamic Storage Allocation Scheme. Computer Journal. Vol. 5, No. 3, 200-209 (1962).
4) Iliffe, J. K.: Elements of BLM. Computer Journal, Vol. 12, No. 3, 251-258 (1969).
5) Organick, E. I.: The Multics System. The MIT Press (1972).
6) Fabry, R. S.: Capability-Based Addressing. CACM, Vol. 17, No. 7, 403-412 (1974).
7) Dennis, J. B. and Van Horn, E. C.: Programming Semantics for Multiprogrammed Computations. CACM, Vol. 9, No. 3, 143-155 (1966).
8) Ackerman, W. B. and Plummer, W. W.: An Implementation of a Multiprocessing Computer System. In Proc. of SOSP (1967).
9) Wilkes, M. V. and Needham, R. M.: The Cambridge CAP Computer and its Operating System. North Holland (1979).
10) Lampson, B. W. and Sturgis, H. E.: Reflections on an Operating System Design. CACM, Vol. 19, No. 5, 251-266 (1976).
11) England, D. M.: Architectural Features of System 250. In Infortech State of the Art Report on Operating Systems, Infortech (1972).

12) 土居範久：オペレーティングシステムのセキュリティ．土居範久・小山謙二編コンピュータ・セキュリティ，共立出版 (1986).
13) Denning, D. E.: Cryptography and Data Security. Addison-Wesley (1982).
14) Harrison, M. A., Ruzzo, W. L. and Ullman, J. D.: Protection in Operating Systems. CACM, Vol. 19, No. 8, 461-471 (1976).
15) Jones, A. K.: Protection in Programmed System. Ph. D. thesis, Carnegie-Mellon Univ. (1973).
16) Wulf, W. A., Levin, R. and Harbison, S. P.: HYDRA/C.mmp: An Experimental Computer System. McGraw-Hill (1981).
17) Jones, A. K.: Protection Mechanisms and Enforcement of Security Policies. In Lecture Notes in Computer Science 60, 228-251, Springer-Verlag (1978).
18) Organick, E. I.: A Programmer's View of the Intel 432 System. McGraw-Hill (1983).
19) Berstis, V.: Security and Protection of Data in the IBM System/38. Proc. of 7th Symp. on Comp. Arch., 245-252 (1980).
20) Houdedek, M. E.: Soltis, F. G., and Hoffman, R. L., IBM System/38 Support for Capability-Based Addressing. Proc. of 8th Symp. on Comp. Arch. (1981).

(昭和62年2月2日受付)