解　説

情報保護の機構*

和田英一**

1. 情報保護の背景

論語に“過猶不及也”という．この論法をもってすれば情報の過処理は処理ではない．過処理即ち，処理しないでよいものまで処理すること，あるいは処理してはならないものまで処理することは，慎むべきである．慎しみによりうまくゆく，これは大変結構なのだが，人には過ちもあり悪意もあり，安寧秩序ある情報処理システムには，過処理を防ぐための保護機構，それも過保護にならぬものが必要となる．

保護機構，これは一種のブレーキである．遅いくるまが簡単なブレーキでよくても，高速なくるまには高性能なブレーキが必要なように，情報処理にも，処理能力のレベルに応じて，いろいろな保護機構が，光が強くなれば，おのずから陰も濃くなる関係に，考案されてきた．この歴史を概観するまえに，情報保護の意味を簡単にのべておく．

情報保護とは前述のように過処理を防ぐものだが，別のいい方では許されぬアクセス (unauthorized access) を防ぐものである．これは許されぬ人によるアクセス，許されぬ仕方によるアクセスを含む．勿論記憶保護，つまり書くことを許されぬところに書く，データを変更する，データをこわす，ことからの保護を含む．また機密保護，つまり読むことを許されぬデータを読む，データがあるかないかを知る，ことからの保護を含む．さらにプログラム保護，つまりプログラムが，許されぬプログラムから，許されぬ時に，許されぬパラメータで実行されることからの保護を含む．保護機構の設計としては，最悪の事態に備えなくてはならないから，記憶保護では，保護を記憶への書き込みにより侵そうとした侵入者に，その成否がわからない場合も含める．プログラムエラーにより間違った場所に書き込むのはこの好例である．機密保護では，直接データが読みだせなくても，正当なプログラムのそのデータへのアクセスのパターンが観測できて，それからデータがなにらしいか侵入者に推測のつく場合も含める．また侵入者には成否がわからなくても，情報保護を犯した結果，以後正当な利用者の正当なアクセスのできなくなるようなことを防ぐことも含める．これは情報保護のために必要な情報，例えばパスワードの表をこわされたというような場合なら記憶保護の問題であり，他人のパスワードが読めたので，その人のデータを全部消去したという場合なら機密保護の問題である．

こんなことを扱うのが情報保護だから，これではうまくいって当り前，うまくいかない時だけ文句をくうという，この世の中にも沢山例のある損な仕事のひとつである．また誤り易い人間や，こわれ易いハードウェアも相手にしなければならないと，何が何だかわからなくなるから普通はいろいろな仮定をおいて，その中で保護機構が完璧に働くようにし，不確定要素に対しては，ある程度のことを考え，あとは保険の考えで諦める．

さて初期の計算機には，どんなことでもできる万能計算機であることを標榜していた程だから，情報保護の考えはあまりない．また配線盤プログラムはこわされる心配はなかった．しかし人間の過ちを予防する意味での入力ルーチン，サブルーチン，アセンブリールーチン等の EDSAC の方針は情報保護と無縁ではない．情報保護のうち記憶保護は，磁気テープの書き込み用リングが最初かと思われる．モニターによる連続処理とわりこみ機能の出現は，コアに常駐するモニターを守るための記憶保護方式の考案に至る．これだけでモニターは安全になったが，すぐつづいてプログラム保護の装置のついた計算機も登場した．機密保護の必要はタイムシェアリングシステムの登場で促進された．タイムシェアリングシステムの実用化には，多勢の利用者のデータをファイルとして保存しておくため自分のデータやプログラムを他人に使われないことの

* Mechanism of Information Protection by Eiiti WADA (Faculty of Engneering, University of Tokyo).
** 東京大学工学部計数工学科

保証をしなければならなかったからである．データの持ち主である証明のためにパスワード方式が工夫されたのもこの頃であった．

タイムシェアリングシステムの成功はつぎにデータをシェアするコンピューターユーティリティやデータベースの構想に発展する．この方式では人によって同じデータでも異なるアクセスができる必要があった．Saltzer の教科書にある例を借用するなら，飛行機の座席予約装置・予約申込みはどの窓口からでもできる．しかし取り消しは申し込んだ窓口でしか出来ない．また窓口では申し込み客，搭乗客の名簿を出力してみることはできないが，乗務員は自分のフライトの搭乗者名簿を離陸直前に限りリストすることができる．

一方或る教科目の学生の成績のファイル．学生は自分の成績だけ見ることができる．演習担当の助手は自分で受け持つ学生の成績をつけ，見ることができる．そして，教科目担当教官は全員の成績を見ることができ，申し出があれば，成績を変更することができる．この2例はあるデータに対しそれぞれのカテゴリーの人が読む，書くのいずれが可能かの組み合わせをきめることで実現できるが，さらにきめ細かい制御も考えられている．つまり利用者がアクセス制御を自由に指定できるシステムで，これにより，販売プログラム (proprietary program) の実現を可能にしようとするものである．

販売プログラムとは，ある開発者の作ったプログラムを利用者が使用しては開発者に使用料を払うもので簡単にいえば現在，メーカーの作ったコンパイラーを使用している利用者が，使用料を払うようなものと思えばよい．現在ではコンパイラーが何回使われたかわからないから，販売プログラムにならないし，メーカーのコンパイラーを一回コピーして，複製の方を何回も使った場合とか，多少手を入れて，これはメーカー製のコンパイラーとは既に異なる，といわれた場合とか，むずかしいことが沢山ある．またコンパイラーの方が，コンパイルするついでに利用者のデータを読んだとか変更したとか，要するに効能書き以上のことをやらない保証も必要である．M. I. T. の Multics[24] や C. M. U. の Hydra などの操作システムは，この程度の情報保護まで狙っていろいろな工夫をしているけれども，まだむずかしいようである．

以上は計算機に直接関係した情報処理の要求とそのための保護の例であるが，計算機外の一般社会にも処理と保護の対応が沢山みられる[10],[33]．通信事業が発達してくれば憲法が「通信の秘密は，これを侵してはならない」と釘をさす．世の中になにに情報システムというのが出現し，計算機網で処理能力を高めてくれば，スウェーデンのデータ法，アメリカのプライバシイ法[32]の制定をみる．日本では行政管理庁が諮問していたが「時期尚早」ということになった[44],[45]．

このような情報保護のために考えられていることを計算機外と内の問題に大別して以下にのべよう．

2. 計算機外の機構

i. 身元確認

計算機システムに対してだけでなく，もっと一般的に，アクセスの制御には，合い言葉と鍵が使われてきた[37]．どちらも記号列だが，前者はよりロジカル，後者はよりフィジカルである．合い言葉では「山」と「川」が有名だが[40]，Hamlet の1幕1場「国王陛下万才！」や Lear 王の4幕6場「マヨナラの花」も合い言葉である．Ali Baba の「開けゴマ」が合い言葉か呪文かは別として，まだまだ例はある．合い言葉は他人に知られたら役にたたない．そこでタイプライターの印字抑制や，オーバータイプで読めなくする方法，乱数を使った方法などが工夫されている．

Project MAC での経験では，コンソール室で或る利用者がログインしようとするとその部屋にいた他の利用者は何気なく席をはずして合い言葉をタイプするのさえ見ないようにしようとしていた．また Multics には比較的覚え易くかつ適当にランダムな合い言葉を作ってくれるプログラムがあるという．他人の合い言葉がどうすれば盗めるかというのは面白い思考実験で，システムに偽装する (masquerade) のはどうかという話がある[30],[38]．また，端末にミニコンをつけて次々と合い言葉を発生するというのも当初は考えられたが，この方法に対してはシステムは防衛できる．

鍵または ID カードはこの読み取り機が普及すれば計算機システムでの身元確認にもっと使われるであろうが，鍵は落とす心配がある．身体についているものなら落とさないというので，声紋や指紋はどうかという考えもあるが，余程高価な情報を保護するのでなければ，このシステムが高価になって採算がとれないであろう．筆者は今のところキャッシュディスペンサーのような合い言葉と鍵の組み合わせが一番実用的かと考えている[43]．

ii. 計算機室の施錠[1]

数年前の大学紛争で，京大の計算機センターには，

「叛」の字がスプレーで書かれ，屋上にヘルメット姿の監視が立ったりしたが，幸いにもセンターの情報がやられたということは聞かない．アメリカでは計算機室が襲撃されたことがあるそうだ．昔，池袋の西武百貨店の火事で計算機室が被害をうけたように記憶しているし，最近は新聞によると間組の爆弾事件が計算機の近くであったようである．

保護すべき情報は計算機室（かその近く）にあるのだから，直接ここがやられないようにしなければならない．それには計算機室がどこにあるかを発表しないことが第一（それでも空からファントムジェット機が落ちてくることもある），それには昔の築城のあとのように工事に関与した人を全部殺してもしなければ，吉良邸の如く敵に知られてしまう．敵の知らないことに頼る保護は，或る条件の中で最悪の事態を考える情報保護では，まず問題にならないから，計算機室には厳重に鍵をかけ，なるべく人を入れないようにする．Kissinger 家のゴミ袋を頂戴した新聞記者がいる位だから，計算機室から出るものについては，ゴミ，紙，リボン，磁気テープ，計算機，人，水，煙，光，音，電波，なにに対しても超敏感になる必要がある．計算機室の方でださなくても，隠しマイク，隠しカメラでとられているかも知れないから，それらの設置されそうなものはおかない．Beardeley の解説の面白い話に「隠しマイク探し屋」を計算機室に入れたら，マイクがなかっただけでなく，探知器のおかげで磁気テープが駄目になっていたというのがあるから[2]，最新鋭の機械の持ち込みもご遠慮願おう．この方は実は考えはじめたらきりがない．計算機のシステム程の完璧さは仲々得られないから，この辺でやめておき，あとは読者におまかせする．

3. 計算機内の機構

i. 方　式

情報保護用方式の最初は，記憶保護のものである．書き込める記憶装置のほかに，読みだす一方の固定記憶装置を持った計算機もあったが，割り込み方式にやや遅れて登場した記憶保護方式は，まず計算機のプログラムの実行にふたつのモード，すなわちモニターモードとユーザーモードを設ける．モニターモードはユーザーモードからみると一種のハードウェアであってモニターモードでは命令が何でも実行できるのに対しユーザーモードでは，書き込み可能領域を示すレジスターで指定される範囲以外への書き込みはできないようになっている(図-1)．万一書こうとすると割り出す．領域レジスターは命令で書きかえるのだが，ユーザーモードの　ログラムが勝手にこれを書きかえ，領域を変更してから元々禁止だったところに書けるようになったのでは，手落ちなので，レジスター書きかえはユーザーモードではできない．これも万一やろうとすれば割り出す．

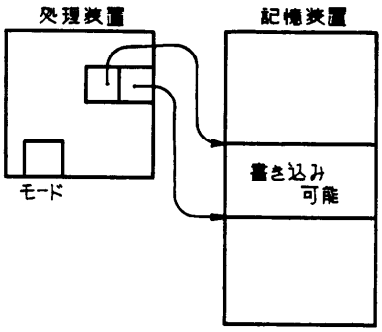

図-1

入力装置から記憶装置に読み込んで，書き込み禁止を侵すことのないよう，入力制御にも領域レジスターを設けるか，入出力命令をユーザーモードで使うと割り出し，割り込み処理プログラムが，この入力命令を解析し，禁止領域に読み込むようなことがなければ入力命令をだす方法がある．割り込み処理プログラムはモニターモードで走っているので，入力命令を実行しても割り出すことはない．

この一番簡単な方式には，ユーザーモードで書き込み禁止領域のプログラムへジャンプしても割り出さないから，実行禁止になっていないが，もしジャンプ先きのプログラムが，禁止領域内に何か書こうとすると，そこで割り出す．しかしもうどこで禁止領域にジャンプしてきたのかわからないから，禁止領域へのジャンプでも割り出す方がディバッグには重宝である．そうすると書き込み禁止領域と実行禁止領域を同一にする必要もないから，夫々のレジスターを用意するのが便利ということになった．レジスターをふたつ，つまり禁止領域をふたつにする位なら，きまってもっと沢山にしようといい出す人がいて，むしろ領域は右折可の如く可能な方を示すことにしよう．ついでに読み出し可能も作ったらということになって，図-2 (次頁参照)に示すような方式が考案されるに至る．ここで R, W, X はそれぞれ読み出し，書き込み，実行の可能を示す．

この方式はこのままで計算機に採用されたかどうか知らない．この方式はひとつにはセグメント方式と一

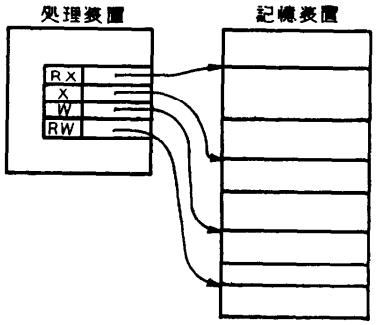

図-2

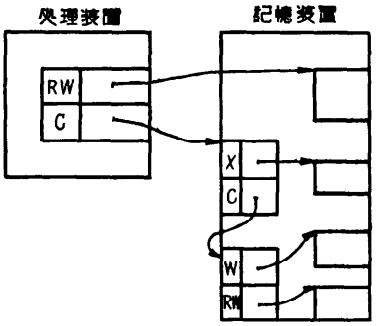

図-3

緒に登場する．すなわちセグメントというのは，プログラムや扱う情報のまとまった単位だから，この単位ごとにアクセス権をつけるのは意味があるし，またセグメント表を毎回参照するのだから，そのついでにアクセス権もチェックできることになる．またレジスターを記憶装置内の表にしたので，長さを気にしなくてよいことになる．もうひとつは記憶装置を固定長のページ単位に分け，領域もページ単位ごとにアクセス用のキイをつける方式に変身した．ページ方式はセグメント方式の下請けになる場合が多くセグメント方式にアクセス権が採用されると，後者の方法は影がうすくなる．

少し前に戻して図-2 のようにレジスターがアクセス権つきでセグメントを示している場合を考える．ここではレジスターの数が限られているから，使えるセグメント数にも限りがあり，それを増やすためにはレジスタを入れかえる必要が生じる．これを勝手にやったのでは書き込み領域レジスターと同じになるのでレジスターの入れかえはモニターモードで行わなければいけないかというと，ここにうまい方法があった．このレジスターに入れるデータは，もともと利用者にアクセスできるセグメントへのものなら何を入れてもいい筈で，それ以外のものでは危くなる．そこでこういうデータは変更しない限り記憶装置とこのレジスター間で自由に移し得る．アクセスに関する情報を変更するためにはアキュムレーターに入れなければならないから，そこでデータをアキュムレーターに入れて変更できるものと領域レジスターとだけ入れ換えがでるだけで情報は変更できないものにする．一方記憶装置もこの2種のデータのどちらのためのセグメントかに大別してしまう．この2種の前者を一般データ，後者をケイパビリティとよぶ．つまりケイパビリティは，ケイパビリティのセグメントとレジスターの間だけ往き来し，アキュムレーターへはもってゆけない．反対に一般のデータはケイパビリティセグメントへ行くことも，領域レジスターに入ることもできない．図-3 で処理装置内にあるのが領域レジスター，Cのアクセス権で指しているセグメントがケイパビリティセグメントである．この方法の大変面白いのは，ひとつのセグメントにケイパビリティ用と一般用のふたつのポインターをつけ，どちらのポインターを使うかによって二重人格をもたせうることである．システムプログラムがケイパビリティを最初に用意するときはこの方法による．以後システムプログラムはケイパビリティを作るに際してこのような危いケイパビリティは決して作らないようにする．ケイパビリティの計算機はイギリスの Cambridge で1台試作中[19]，またイギリスに商用機が一機種あるらしい．あとはバークレーで作っていたがどうなったか．

ケイパビリティと一般データをセグメント単位で分ける方法に対し，記憶装置内の各語に，これはどちらの種類のデータかのタグをつける方式も提案されている[6]．この方法では，ケイパビリティレジスターから記憶装置に入れられたデータにはケイパビリティのタグがつき，アキュムレーターから入れられたものには一般データのタグがつく．アキュムレーターに入れるデータには一般データのタグが，レジスターに入れるにはケイパビリティのタグがついていなければならない．この方式では二重人格タグが作れないからケイパビリティを作るための何らかの方法を講じる必要がある．

タグ方式はデータが自分の性質を常にもち歩いていて，それに見合った操作しかゆるされないことを実現しているのである．簡単な例では整数のデータには整数のタグがついていて，実数の演算を試みると文句をいわれるようなものである．ケイパビリティは番地の

性質をもっているので，うっかり掛け算などやってはいけないという方式と解することができる．

或るセグメントに対するアクセス権は一定かというとそうではないことは，座席予約機や学生の成績ファイルの項でのべた通りである．しかも同一利用者のプログラムが走行しているあいだにもアクセス権は変化する．例えばプログラムをコンパイル＋ゴーするときコンパイル中にはオブジェクトプログラムのセグメントを書き込み可能にしておき，オブジェクト実行中は実行可能にしなければならない．このようにアクセス権が刻々とかわるが，アクセス権の同じプログラムの部分をドメインとよび，上のようなのはマルチドメインの例である．ドメインを高級なものから低級のものまで一列にならべ，或るセグメントに対するアクセス権が，高級なものは低級のものを含むような構造をもっているとき，これをリングとよぶ[34]．モニターモードを高級ドメイン，ユーザーモードを低級ドメインとよべば，ユーザーモードでできることはモニターモードではすべてできるから，これはリング機構といってよい．

リング機構はMulticsの考案で，日本ではHitacの仮想空間の機械に採用されている．この動作に関してはSaltzerとSchroederの報告がある[28]からその流れ図をご覧願いたい．ご覧になればすぐおわかりのように，そう全面的に気持ちのよいものではない．どこが悪いかといえば，ひとつはインダイレクト参照である．つまり命令（というか利用者）のリング（ドメイン）と対象のほかに中継点があり，これをアクセスする側ともされる側とも割り切ることができず，安全サイドに解釈していることにしている．もうひとつは，「実行のみ」のセグメントがなく，「実行のみ」はいつも「読み出し可」のアクセス権も認められること．これはフラグメンテーションのロスに目をつぶって，別セグメントにし，方式を多少変更すればよく，そう本質的ではない．第3は，サブルーチンをコールしたところはいずれサブルーチンから戻ってくるのでゲートにしておく必要があり，サブルーチンから帰ってくると今度は無闇にとびこまれないよう，ゲートでなくなさなければならないのがこの方式にないこと．これはドメイン情報のスタックなどとともに方式を根本的に設計しなおさなければならないであろう．

とにかくいまだに販売プログラムまでうまく実施できる比較的簡単で本質的なアーキテクチャは存在していないようであり，方式屋さんの新しい提案が待たれる．

ii. 操作システム

方式設計がいくら完全なシステムが組み込めるように用意されても，その上で走る操作システムが完全でなければ，システムとしては失敗である．そこで方式設計の次にはちゃんとした操作システム作り方が問題となる．ちゃんとした操作システムではひとつにはシステム全体を沢山のレベルに分け，各レベルで完全性の証明をして，全体としても完全にしようという考えがDijkstraあたりから始まったのはまだ記憶に新しい．THEのシステムはあまり大きくないらしく，このように証明のつみあげでゆけたけれど，もっと大きなシステムにもこの方法が使えるとして，努力を注いでいる人もいる．しかし操作システムは並列度が高いので証明はずっとむずかしい．

「このことは耳にタコができる程いわれてきた．しかし何度くりかえしても決して強調しすぎにはならない」システムの作り方の原則のひとつは，それをシンプルにすることである．しかし悲しいことに人間はすぐ大きくて手に負えないシステムを作ってしまう．特に操作システムはいろいろな要求により，少しずつ変更されてゆくので，そのときに虫が入って，完璧でなくなる恐れがある．

この対策はふたつある．ひとつはいわゆるポリシーとメカニズムの分離で，メカニズムの方だけ完璧に作り，ポリシーの方はそう完璧でなくても，システム全体としては情報保護の点からは完璧になるというもので，C.M.U.のHydraはこのメカニズムの部分，いわゆる核を作ろうというものである．もうひとつは操作システムの設計に構造的プログラミングを採用，それも操作システムはマルチプログラムで走るのが今や常識になってきたので，構造的マルチプログラミングを採用することである．

この分離の考え方も，構造的マルチプログラミングの考え方も，Brinch Hansenあたりが主流で[3]，核を構造的マルチプログラミングで書くという方向へ進んでいる．構造的マルチプログラミングというのは，極く簡単にいうなら，マルチプログラミングの基本操作のセマフォーに対するPやVオペレーションは，gotoに相当する程基本的であり，こんなのをやたらに使ったマルチプログラムはgotoをやたらに使ったプログラムと同様，信頼性は非常に低い．そこでマルチプログラム用のもう少し本質的な操作をいろいろ工夫して

それで完璧な操作システムを作ろうというのである．この考えをとり入れてインプリメント中の言語がConcurrent Pascal[4]であるが，言語の仕様はまだ整理の余地が残っているような気がする．

情報保護は計算機システムのいろいろな分野に関係するが，やはり操作システムが主役のようである．一昨年開かれたACMのOperating System Principleのシンポジウムでも，操作システムとプログラム言語のインターフェースの討論会でも，情報保護が重要テーマであったようだ．情報保護の研究と開発が現在どのように進んでいるかの概説は，今は少し古くなったが昨年夏に出たSaltzerの報告[26]を参照されたい．

iii. プログラム言語

情報保護は操作システムの問題と思われていたところへ，これはまたプログラム言語の問題でもあると指摘したのはMorrisが最初であろうか[17),18)]．

ひとつは前述のようなケイパビリティとは番地であるという解釈からくる．番地，すなわち変数はプログラム言語では型の宣言がおこなわれ，その変数をもっていることは，その型だけのデータを読んだり書いたりできることになる．ケイパビリティはあるセグメント，対象を指すように描くことが多いが，これはポインター，或いはレファレンスであり，Algol 68の変数がレファンスであるという解釈とも一致する．

もうひとつは殆んどのプログラム言語はプログラムの部分の書き直しができず，そういう意味ではアセンブラーのプログラムとちがってすでに一種の保護ができているのではないかという解釈．またブロック構造を持っている言語では変数などにスコープがあって，プログラムにアクセスできる場所とできない場所とがあり，これはマルチドメインのアクセス制御と一脈通じているではないかと言いだすわけだが，これは少しよく考えてみると決して偶然ではない．これは我々が情報保護を必要と考え始めたひとつの要件，すなわちプログラムのエラーを防ぐのに，プログラムでできることにかなり強い制限をつけて実現しようとしたその願望に基づいて，困らない程度に言語を設計したからであった．

かく申す筆者などはトリックができなくなって型の制限をずい分恨むときもある．しかしこれでプログラムの完成は早くなり，プログラムの考え方も規格化されて，ある意味ではプログラムが作り易くなったのはたしかである．情報保護がプログラム言語の問題であることを長いあいだ気づかせなかったのは，処理系の手抜きである．まったくセマンティックスのいう通りにしか実行しない処理系があれば，そしてこの処理系だけしか用意していなければ情報保護は片づいたも同様である（日米コンピュータ会議直前のデータベースの保護の教育セミナーでWilsonはしきりにプログラム言語だけでは保護できないと主張したが，言語の設計と処理系の作り方では可能の筈である）．方式のところでインダイレクトがむずかしいといったのは，これがプログラム言語のセマンティックスと無関係に現れるからどうしてよいかわからないためである．

完全な言語の処理系を作るのは理想なのだが，コンパイル時のチェックはいいとして，ランタイムのチェックは能率を非常におとすので，チェックはできるだけコンパイル時に可能なもののみとし，あとは方式設計の助けをかりるなどという言語設計にするのが，情報保護に関心をおいた場合の方針のようである．

情報保護の観点から新しい願望を満すために提案されたプログラム言語の機構はモニター[4),8),9)]のそれである．これはAlgol 60のようなブロック構造の言語で，ブロックにローカルなオウン型変数が，勿論ブロック外から手が届かないのに対し，ブロック内にブロック外から手の届く手続きを用意し，この手続きからは変数に手の届くのを利用して，外から変数にアクセスしようというものである．計算機の方式で，ある対象に読みだし可能，書き込み可能など，いくつかの標準と思われるアクセス権を用意したわけだが，このに対する要求は次々と複雑になり，とても，はじめからハードウェアの方式に組み込んでおけなくなってくる．それならいっそソフトウェアでチェックすればよいではないか，と誰しも考えつくわけだが，そうだとすると対象に直接手が届いてはならず，必ず手続きを経由する方式になるわけで（Hitac 5020のソフトウェア命令も多分にこの趣向のものであった），この考えのプログラム言語版が，モニター構造ということになる．これはまたデータの型の抽象化として，型の機能の方からのみ定義し，表現は利用者は知らなくてよいとする，Liskovらによるプログラム言語Clu[16]のクラスターの考えにも相通ずる一般性のあるものである．モニター構造は，手続きが外からよばれ，中に手がとどく，つまりブロックの外にあるようで中にもあるような両的存在であるが，ブロック構造言語の処理系でのブロックへの入り方をわずかに拡張することで比較的容易にインプリメントできるのではないかと思われる．

情報保護に関係したプログラム言語の機能はモニターを更に一般化したシールの考えである．モニターでは変数は手続きに依頼して処理してもらったが，ブロックの外に変数を持ちだすことはできない．これを持ちだせるようにもしたいという願望である．どういう場合に必要になるかというと，仮りに誰かが途轍もなくすぐれた複素数のパッケージを作って提供したとする．これは販売プログラム故，ユーザーは中身は知らない．ユーザーは実，虚部を渡して複素数のこのパッケージ表現を作ってもらって変数に代入してもらう．こういう変数をふたつ送って加算してもらうと，和が代入されてくる．ユーザーは変数を勝手に自分の勢力内で移しかえることはできるが，残念ながら中は読むことができないのである．これは丁度，パッケージはトランクに情報を入れてシールして渡してくれるので，利用者はトランクを移動しているだけという連想である．パッケージはアンシールの機能をもっている．こうして販売プログラムの権利を守ろうというのだが，これは部分的には既に実現されていると見られる節もある．中を覗く，これは変数のレファレンスをはがすことである．これはオペレーションだから，或る範囲から先で禁止するなんていうのは難かしい．もうひとつはレコード型でフィールドをセレクトすることである．複素数を R_e と I_m なるセレクトのフィールドでできているとすれば，R_e, I_m でセレクトすれば中が見える．だから先刻のパッケージはセレクターをユーザーに教えなければよい．Algol 68 ではシステム用にアレフという文字のセレクターが特別にとってあって，これで情報保護をしている．アトムになったら中が見えない筈の Lisp でも，セレクター car, cdr を公開したからユーザーでもｐリストが自由によめるようになってしまった．あれはアトムから先は atomcar, atomcdr などとしておいた方がよかったとも思う．要するにセレクターは一種の鍵であり，情報の保護はセレクターに転嫁できる．セレクターを上手に外に輸出してフレキシブルに扱えるよう考えているのは，Wulf らによる Alphard である[31]．

4. 情報保護の展望

今後情報保護はどうなってゆくか．残念ながら予想は困難である．数年前日本でもタイムシェアリングがもっとはやりそうにいわれていたので，期待はずれ物語でとりあげられてしまった[35]．情報保護でいえるのは，この道の技術者として，過保護は慎しまなければならないことで，これは保護がすぎると，ユーザーを失望に追いたて，計算機が使われなくなり，真価が発揮できなくなる．どこからが過保護かをきめるのはこれまた困難で，この春来日していた McCarthy と雑談していたら，かれには Multics も過保護とみえるらしかった．

情報保護の方には最小特権の原則 (minimum privilege の principle) というのがある．これは操作システムはユーザーに対し，リソースもアクセス権も，生かさぬように殺さぬようにぎりぎりだけ与えるのがよいというのである．そのあらわれが，アクセス権は黙っていたら何も貰えず，要求すれば，それがリーズナブルなら，それだけ下賜される，ということになる．これをできるだけきめ細かにやったらどうなるかという思考実験は面白い．或るプログラムが走るとき，各命令の実行が次々と新しいドメインとなってゆく．このドメインでできること，それはこの番地から，実数 3.14 を読みだし，アキュムレータにたいして結果をなにがしにすること，それ以外だったら割り出しが起きる，という具合に計算が進行する．たしかに計算はできるのだが，結果も，あらかじめ最後のドメインで指定されている以外の価では叱られる．となるとなんのために計算したかわからなくなる．これは笑い話ではあるけれど，情報保護の研究はまかりまちがえば，ここに行きつく可能性をはらんでいる．そうならぬよう望むこと切である．

参 考 文 献

(本文に引用してない情報保護関係の文献を含む)

1) AFIPS: AFIPS System Review Manual on Security, p. 109, AFIPS (1974).
2) C. W. Beardsley: Is Your Computer Insecure?, IEEE Spectrum, Vol. 9, No. 1, pp. 67～78 (1972).
3) P. Brinch Hansen: A Programming Methodology for Operating System Design, Inforamtion Processing 74, pp. 394～397 (1974).
4) P. Brinch Hansen: Concurrent Pascal, A programming Language for Operating System Design, p. 44, Information Science Technical Report No. 10, California Institute of Technology, (1974)
5) A. Evans Jr. 他: A User Authentication Scheme not Requiring Security in the Computer, Comm. ACM, Vol. 17, No. 8, pp. 437～442 (1974)
6) E. A. Feustel: On the Advantage of Tagged

Archtecture, IEEE Trans. on C. C-22, No. 7, pp. 644～645 (1974).
7) G. S. Graham and P. J. Denning: Protection—Principles and Practice, '72 SJCC, pp. 417～429 (1972).
8) C. A. R. Hoare: Astructured Paging System, Comp. J., Vol. 16, No. 3, pp. 209～215 (1973).
9) C. A. R. Hoare: Monitors: An Operating System Structuring Concept, Comm. ACM, Vol. 17, No. 10, pp. 549～557 (1974).
10) L. J. Hoffman: Security and Privacy in Computer Systems, p. 422, Melville Publishing Co. (1973).
11) I. B. M.: Data Security and Data Processing, Vols. 1, 2, 3-1, 3-2, 4, 5, 6 (1974).
12) A. K. Jones: Protection in Programmed Systems, p. 139, Department of Computer Science, Carnegie-Mellon University (1973).
13) B. W. Lampson: Protection, Operating System Review, Vol. 8, No. 1, pp. 18～24 (1974).
14) R. R. Linde: Operating System Penetration, 1975 AFIPS, pp. 361～368 (1975).
15) B. H. Liskov and S. Zilles: Programming with Abstract Data Types, SIGPLAN Notices, Vol. 9, No. 4, pp. 50～59 (1974).
16) J. Martin: Security, Accuracy and Privacy in Computer Systems, p. 626, Prentice Hall (1973).
17) J. H. Morris, Jr.: Protection in Programming Languages, Comm. ACM, Vol. 16, No. 1, pp. 15～21 (1973).
18) J. H. Morris, Jr.: Authentication Tags—The Proper Division of Hardware/Software Responsibility, (Unpublished Memo, Univ. of Calif. at Berkeley).
19) R. M. Needham: Protection Systems and Protection Implementations, 1972 FJCC, pp. 572～578 (1972).
20) G. J. Popeck: Protection Structure, Computer, Vol. 7, No. 6, pp. 22～33 (1974).
21) G. B. Purdy: A High Security Log-in Procedure, Comm. ACM, Vol. 17, No. 8, pp. 442～444 (1974).
22) D. D. Redell: Naming in Protection in Extensible Operating Systems, p. 161, MAC TR-140, MIT Project MAC (1974).
23) L. J. Rotenberg: Making Computers Keep Secrets, p. 393, MAC TR-115, MIT Project MAC (1974).
24) J. H. Saltzer: Protection and Control of Information Sharing in Multics, Comm. ACM, Vol. 17, No. 7, pp. 388～402 (1974).
25) J. H. Saltzer: Ongoing Research and Development on Information Protection, Operating Systems Review, Vol. 8, No. 3, pp. 8～24 (1974).
26) J. A. Scherf: Computer and Data Security: A Comprehensive Annotated Bibliography, MAC TR-122, MIT Project MAC (1974).
27) M. D. Schroeder: Cooperation of Mutually Suspicious Subsystems in a Computer Utility, p. 167, MAC TR-104, MIT Project MAC (1972).
28) M. D. Schroeder and J. H. Saltzer: A Hardware Architecture for Implementing Protection Rings, Comm. ACM, Vol. 15, No. 3, pp. 157～170 (1972).
29) M. J. Spier, T. N. Hasting and D. N. Culter: A Storage Mapping Technique for the Implementation of Protective Domains, Software—Practice and Experience, Vol. 4, No. 3, pp. 215～230 (1974).
30) M. V. Wilkes: The Cambridge Multiaccess System in Retrospect, Software—Proctice and Experience, Vol. 3, No. 4, pp. 323～332 (1973).
31) W. A. Wulf: Alphard: Towards a Language to Support Structured Programs, Computer Science Department, C. M. U. (1974).
32) Privacy Act of 1974 Becomes Law, Comm. ACM, Vol. 18, No. 2, p. 138 (1975).
33) アーサーミラー(片方訳): 情報とプライバシー, p. 368, ダイヤモンド社 (1974).
34) 石田晴久: 超大型コンピュータの技術(4), 情報の機密保護技術, bit, 1974年1月号.
35) 石田晴久: コンピュータ期待はずれ物語——日を風びしなかった TSS, bit, 1975年6月号.
36) 奥平康弘: 情報化社会とプライバシーを考えるために, 朝日新聞, 1975年4月7日.
37) System-5: 計算機に対する「合い言葉」, 数学セミナー, 1974年9月号.
38) System-5: 計算機システムでの情報保護, 数学セミナー, 1974年10月号.
39) 情報処理学会: Tutorial for Data Base Protection and Security テキスト, p. 298, 情報処理学会 (1975).
40) 高橋博: 話し言葉——合言葉, 朝日新聞, 1975年8月23日.
41) 松浦太郎: 電算機をガッチリ守ろう, 科学朝日, 1973年8月号.
42) 吉村鉄太郎: 情報システムの機密保護, bit, 1973年12月号, 1974年1月号.
43) 和田英一: データベースの機密保護, 昭和50年電気四学会連合大会予稿集 (1975).
44) 電算機とプライバシー, 朝日新聞, 1975年4月10日.
45) 電算機とプライバシーの保護, 朝日新聞, 1975年4月11日 (社説).

(昭和50年9月4日受付)