解　説

仮想記憶方式†

武井　欣二††

1. はじめに

現在，仮想記憶は広く実用されているので，個々の計算機システムに具体化された形でのイメージは，もはや解説を必要としない程度に，普及されていると考えられる．したがって，ここでは解説における通常のアプローチとは逆に，概念を具体的なものから抽象的なものに一般化する方向をとり，仮想記憶方式をできるだけ広範なパースペクティブにおいて捉えること，各方式の原理的な差異を明確にすること，そして，できうれば，仮想記憶の発展可能性がどの辺にありそうかを，多少なりとも示唆しうるものであることを目標にしている．

[注] 完結した仮想記憶方式の解説としては，当然触れておかなければならない話題のいくつかが，ここではまったく無視されてしまっている．それらのうちの一部のもの，たとえば，仮想記憶の歴史的な側面とか，制御アルゴリズムについては，本特集の他の解説との重複を避けたためであるが，中には，各方式の実現形式などのように，実在のシステムごとの個別性が強いものは，意識的に逃げる方針をとったことによって，洩れてしまったものもある．

2. 基本概念

2.1 仮想記憶の原理

仮想記憶ということばが，現実にどのような意味で通用しているのかを見極めることは，大変に難しい．特に，その定義に一応の明示性と厳密性を求めることになると，なおさらである．実在のものでの代表例の一つをあげると，“仮想記憶とは，一つのアドレス空間であって，その最大容量が，計算機システム内に存在する物理的なプロセッサ記憶にある記憶位置（ロケーション）の実際の個数ではなくて，そのシステムでサポートされているアドレッシング機構によって定められるようになっているもののことである[1]”．こうした定義をやや抽象化して，ここでは次のような定義を用いることにしたい．すなわち，“仮想記憶とは，主記憶に与えられている論理形式に等価変換された形で利用されるようになっている．二次記憶のことである”．この定義の利点は，まえのものより多少直観的になっていることと共に，仮想記憶機構の本質が等価変換操作にあることが，明確に表面に出ている点にある．そして，この等価変換において，不変のまま保存されなければならないのは，主記憶に与えられた論理形式の中に盛られている限りの記憶装置の性質であって，それ以外の性質は直接には無視されてしまうことも，そこに含意されているのである．この意味では，仮想記憶は一つの抽象化になっているといえる．さらに，この‘主記憶に与えられた論理形式’そのものからして，すでに実際の物理的な存在としての主記憶装置からの，何段階かにわたる抽象化によって与えられるものである点にも注意が必要である．最後に，この定義には，仮想記憶の容量が二次記憶のそれに依存することも含意されている．しかし，一般的な記憶階層の立場からいえば，ここにいう二次記憶が必ずしも物理的な実在としての二次記憶装置を意味するものとして，限定して考える必要がないことはいうまでもない．

仮想記憶は，二次記憶を主記憶に変換するものであるという意味では，記憶階層内の一部分のみを扱う，局所的な方式ということになるが，ここにみられる変換方式には，記憶階層内の他の階層間にも応用が効く考え方が多くみられる．しかし，以下では記述を簡単にするために，この点をいちいち指摘することはしない．その代わり，記述そのものをできる限り一般化，抽象化することにする．たとえば，うえでの仮想記憶の定義において，主記憶とあるところを n 次記憶，二次記憶とあるところを $n+1$ 次記憶に，それぞれ置換えれば，それは一般的な記憶の仮想化を定義するのにも利用できるかも知れない．

2.2 主記憶の論理形式

† Virtual Memory Schemes by Kinji TAKEI (NEC-Toshiba Information Systems Ltd.),

†† 日電東芝情報システム(株)システム技術部

機械語の水準でみた，データや命令の形式と，それらを主記憶上で指示するのに用いることのできる名前全体の集合とを一緒にしたものを，主記憶の論理形式という．

現在の常識的な計算機システムにおいては，主記憶は，語またはバイト（時にはビット）から成る1次元配列としてのイメージで扱われ，データや命令は，その配列内の連続した何語または何バイトかを占める形式を与えられているため，それらを指示したいときには，配列内でのその先頭の語またはバイトのインデックスを用いることができる．このような場合には，主記憶の論理形式は，語，バイトなどの形式とその配列に対するインデックス全体ということになる．さらに省略を進めると，語，バイトなどの形式は暗黙の前提として無視し，専らインデックスだけを問題にすることになる．このような意味においてのインデックスを特に実番地とよび，実番地全体から成る集合のことを実記憶，実空間，実番地空間などという．

実記憶の概念は，もともと，実際の物理的な主記憶装置（プロセッサ記憶ともいう）に内在している論理形式を抽出したものとみることができる．これに対して，等価変換によって二次記憶上に展開されることになる主記憶論理形式のことを，仮想記憶，仮想空間，仮想番地空間，あるいは単に番地空間などとよぶ．そして，そこでのインデックスのことを，論理番地，仮想番地，あるいは，ただ単に，番地などという．

2.3 等価変換の機構

二次記憶を主記憶に与えられている論理形式に等価変換するための，最も単純な機構は，二次記憶と同じだけの容量をもった主記憶を用意しておき，二次記憶にアクセスする必要が生じた場合には，二次記憶の内容をそっくり主記憶に移してアクセスし，必要がなくなったところで主記憶の内容を二次記憶に戻しておくものである．いうまでもなく，この方式は効率が悪くて非現実的であるが，主記憶を時分割的に多重利用している点では，記憶階層としての合理化の方向に沿ったものということができる．

さらに合理化を進めるための着眼点は，実際に行われるアクセスに観察される，次のような事実である．すなわち，二次記憶の内容は全体がいつも一様にアクセスされているのではなく，ある時間幅だけについてみると，その間に本当に参照される内容は，全体の中のかなり局限された部分のみに集中する傾向がある．この事実は‘参照の局所性’ということばで引用されることが多い．参照の局所性を考慮すると，いつも二次記憶の内容全体を主記憶にもってきておくのは無駄で，プログラムの進行に従って，必要な部分だけを順次移してくるようにすれば，それらを収容するために必要とされる主記憶は，非常に少なくてすむことになる．ただ問題は，‘必要な部分’ということが，具体的に何を意味するかである．まず，‘部分’というからには，二次記憶の内容をいくつかの部分に区分化して，それぞれを指示できるようにする機構が要る．つぎに，与えられた時点で‘必要な’部分がどれであるかを知る機構がなければならない．

区分化の機構については，基本的な考え方として，二つの方式がある．一つは，二次記憶上の内容がもっている論理的な構造，たとえばサブルーチン構造などに則して区画を切っていくものである．これは通常セグメンテーションとよばれている方式で，これで作られるそれぞれの区画をセグメントという．もう一つの方式は，二次記憶上の仮想空間を，その内容とは無関係に，固定長の連続区間に機械的に区分化するものである．これはページング方式とよばれている．それぞれの区間を書物のページに見立てているわけである．以上の説明で明らかなように，セグメントではそれぞれの大きさがばらばらになるのに対して，ページではそれが一定になっている．

残る問題は，どの部分（すなわちセグメントあるいはページ）が必要であるかを知る機構であるが，これに対しても，基本的に二つの考え方がある．その一つは，プログラムの動作を事前に何らかの方法で推定し，それに従って，どの部分をどういう順番でもってくればよいかを，予めスケジュールしてしまうという考え方である．この方式の問題点は，プログラムの動作を事前に推定しなければならないことである．これをプログラムの作成者にやらせるのが，いわゆるオーバレイである．これは明らかに等価変換の趣旨に反することになる．この推定をシステム側だけで，自動的に行うのは非常に困難である．もう一つの考え方は，その部分内の命令あるいはデータに対するアクセス要求が実際に発生したことをもって，必要性の判定としようというもので，一般にオン・デマンド方式とよばれている．この方式は，割込み機能を利用して，まったく機械的な方法で実現できる．

2.4 番地変換

仮想記憶としての二次記憶上に存在しているセグメントあるいはページの内容には，それぞれ固有の論理

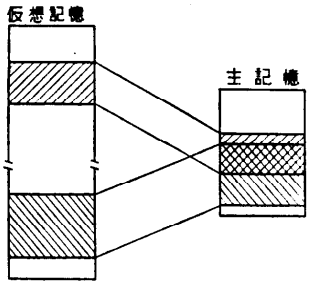

図-1 論理番地と実番地の対応

番地が与えられていることは，第2.2節に説明した通りである．そして，プログラム内部からそれへのアクセスには，これらの番地がそのまま用いられる．このアクセスが実行されるのは，いうまでもなく主記憶上においてであるが，主記憶を実際にアクセスするのには，必ず実番地によらなければならない．図-1にみられるように，一般に論理番地と実番地が一致するようにすることは不可能であるから，アクセスが正しく行われるためには，各論理番地をそれに対応している実番地に変換するための機構が必要になる．これを番地変換機構という．

番地変換機構の基本的な機能は，プログラム再配置に伴う変換の場合と同じである．仮想記憶の番地vから始まるセグメントあるいはページが，主記憶上の実番地rから始まる部分に入って実行されるということは，このセグメントあるいはページが，いったん仮空の実番地vに移され，そこから実際の実番地rに再配置されたものとみることができるからである．したがって，番地変換機構は，いわゆる再配置レジスタと番地計算機構の組合せとして，具体化できることになる．ただし，再配置レジスタは，主記憶上に同時に収容できるセグメントあるいはページの個数と同じだけ必要になる．このため，全部の再配置レジスタは一つの表としての形式にまとめられる．表の各エントリには，互に対応する論理番地と実番地が含まれる．この表を番地変換テーブルという．

2.5 記憶管理

仮想記憶上のセグメントあるいはページは，実際には，主記憶または二次記憶のどこかに存在場所を与えられている．セグメントあるいはページに対する，物理的な記憶領域の割付けに関する制御を，仮想記憶のための記憶管理という．記憶管理には，主記憶を対象にするものと，二次記憶を対象にするものとがあり，両者間には性格的に大きな相異があるため，それぞれ個別的に扱われることが多い．

主記憶管理の中心課題は，アクセス要求が出たとき，必要なセグメントあるいはページが，主記憶上に存在している割合をできるだけ高くすることにある．この課題は，一般に，つぎのようにブレークダウンされている．

・置換えアルゴリズム——主記憶上に新たに空き場所を作る必要が生じたとき，どのセグメントあるいはページを追い出すか，また，このように置換えられることになったものを，いつ，どのようにして追い出すかを定めるもの．

・配置アルゴリズム——主記憶上に複数個の空き場所がある場合，二次記憶からもってくるセグメントあるいはページを，それらのうちのどこに入れるべきかを決定するためのアルゴリズム．

・取込みアルゴリズム——どのセグメントあるいはページを，いつ二次記憶から主記憶にもってくるかを決めるアルゴリズム．

一方，二次記憶管理における中心課題は，主記憶との間に行われるセグメントあるいはページの転送による遅れが，システム・パフォーマンに与える影響をできるだけ小さくすることである．このための着眼点の第一は，転送に要する時間を最小化することである．最小化のための方法は，二次記憶として用いられる補助記憶装置の性質に依存するところが大きい．一般に，補助記憶装置は乱アクセスになっていないので，そのアクセス時間が大きな比重をもつものになることが多い．特に，可動ヘッド・ディスクを利用する場合には，シーク時間が重大な遅れ要因となる．

アクセス時間は，二次記憶に対して相次いで読み書きされる，いくつかのセグメントあるいはページの位置関係によって支配される．この観点から，二次記憶管理方式はつぎのように分類されている[2]．

・固定割当て方式——各セグメントあるいはページに対して，ジョブ実行開始から終了まで常に同一の場所を割当てる．

・浮動割当て方式——各セグメントあるいはページに対して，その時々の状況に応じて適当な場所を，動的に割当てる．

浮動割当て方式は，特殊な場合として固定割当て方式を含んでおり，したがって，それよりもよい効率を与えるものである．しかし，管理という点では固定割当て方式の方が簡単である．

3. 目的と効果

仮想記憶の目的は，直接的には，プログラマからみた主記憶を，物理的な主記憶装置のもつ番地的ならびに容量的な制約から独立させること，すなわち，論理番地を物理番地から分離するとともに，プログラムで実際に利用できる番地空間の大きさを，論理的な限界いっぱいに拡張することにある．

この分離と拡張によって，具体的にどのような効果が期待されるか，いいかえれば，仮想記憶の利用目的とでもいうべきものの概要を以下にあげる．ただし，これらそれぞれの有効性は，計算機技術全体の流れの中で，さまざまな消長を見せるものであることに注意する必要がある．

（1） プログラミング上の効果

・主記憶容量の制約からくるプログラム設計上のオーバヘッド，たとえば，オーバレイの工夫，外部記憶装置への入出力の工夫などが不要になる．

・主記憶容量への配慮によってプログラムの構造が不自然になる，といったことがなくなる．たとえば，最初から大きな構想の下に，オープン・エンディッドな設計を行うことが可能になる．これはプログラムの保守という面でも大きな効果をもたらすものである．

・性能計測手段，デバッグ手段などの補助手段の組込みが，相当自由に行えるようになる．

・既存プログラムの機能拡張が容易になり，さらに，外部とのプログラム交流での致命的なネックも取除かれることになる．

・小規模なシステムにおいても，高度なプログラム開発手段，たとえば，高度な機能をもった高水準プログラミング言語などを，活用できる．

（2） システム運用上の効果

・システム間の互換性が大幅に改善される．たとえば，主記憶容量の小さなシステムで，それの大きなシステムのバックアップをすることが可能になる．

・システム性能が改善される．第一に，多重プログラミングが高度化できるので，それからの効果がさらに向上する．緊急度の高いジョブの割込みや診断プログラムの実行などの影響も最少限になり，さらには，主記憶の一部に障害が発生しても，ジョブ側にはほとんど感知されないといったことも可能になる．

・オペレータの負担が軽減される．

4. 制御方式

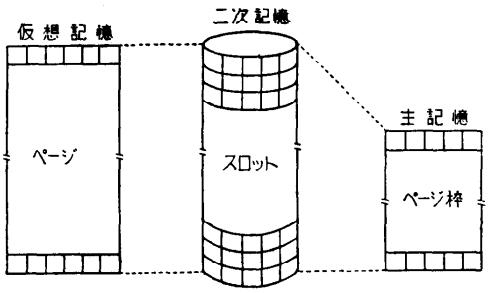

図-2　ページング方式

4.1 ページング方式

（1） 原　理

第2.3節に述べたように，ページング方式は，仮想空間を固定長の連続区間に区分化するものである．ただし，この区分化はプログラマには透明である．これに対応させて主記憶と二次記憶も同様に区分化する．主記憶に生ずる区間のことをページ枠(フレーム)，二次記憶に生ずる区間のことをスロットと呼ぶ．

仮想空間の各ページには，図-2にみられる通り，一対一にスロットが割当てられ，ページの内容がそこに保持されるようになっている．ページの内容にアクセスするために，それをいったん主記憶に移さなければならない．この場合には適当なページ枠が用意されて，そこにスロットからの転送が行われる．この転送動作をページ・インとよぶ．これに対して，ページ枠からスロットへの転送動作をページ・アウトという．

仮想空間，したがってページの内容に対するアクセスは，すべて仮想番地によって行われる．仮想番地は，番地変換機構によって，まずそれが属しているページが主記憶に入っているかどうかが検知される．入っていれば，さらに対応する実番地への変換が行われる．入っていなければ，それをページ・インするための手続が実行される．まず，主記憶管理機構によって，空きページ枠の有無が調べられる．空きが無い場合には，与えられた置換えアルゴリズムに従って，適当なページ枠が選択され，その内容をページ・アウトして空きを作る．このようにして空きページ枠が確保されると，そこに必要なページ内容をページ・インすると同時に，それに対応して番地変換テーブルを更新する．そして，この更新された表をもとに，番地変換機構による実番地への変換が実行される．こうして，目的とするアクセスのための実番地が整うことになる．

空きページ枠をチェックしたとき，場合によっては

複数の枠が見付かることがありうるが，それらのうちのどれを選ぶかは，ページの大きさが一定であり，主記憶が乱アクセスであることから，基本的には問題になることはない．

（2）　ページ長

番地が二進表現をとっている場合には，ページの大きさは2のベキ乗が用いられる．そうすることによって，仮想番地はページ番号とオフセット（ページ内での語ないしバイトの位置）という二つの部分に分割されることになり，番地変換機構ではページ番号だけを変換すればよいことになる．

ページ長は，一般に 1K～16K バイトの範囲で選択されることが多い．ページ長の選び方によってシステム性能が大きく左右されることは確かであるが，これらの間にどのような関係があるかは，関連するパラメータが多くて一般的な議論をすることは困難である．特に多重プログラミングの環境下では，問題が複雑になる．また，プログラムの性質によっても影響され，たとえば，その手続部とデータ部とでは最適な大きさが異なり，データ部の場合には小さくなるほど効率がよくなるという結果が得られている[4]．手続部の場合には，ある部分が参照されると，引続いてその近辺の部分も参照される可能性が高いという性質があり，したがって，ある程度の範囲を一度にもってきておいた方が効率がよい，すなわち，ページ長はあまり小さくない方がよい．ページ長が大き過ぎれば，ページ内に不要な部分をかかえ込む割合が増して効率を下げることになる．実際のシステムの中には，ページ長を一つの値に固定するのを避け，いくつかの値を選択的に切換えることができるようにしている例が多い．しかし，最近の機種に，まえには選択的になっていたのを逆に固定化したものがあるのは注目される．

（3）　番地変換機構

番地変換機構の主な機能は，プログラム実行中に発生する仮想番地を対応する実番地に変換することである．仮想番地の発生源は，1) 命令カウンタ，2) 命令語，3) 入出力コマンド語であるが，これらのうち，1) と 2) の場合はプロセッサで，3) の場合は入出力プロセッサ（ないしチャネル）で番地変換を行うのが通常のやり方になっている．ただし，3)の場合は変換の頻度が低く，それほどの高速性を要しないことから，変換機能をプログラムの形で実現する方法が用いられることもある．

番地変換のためには，仮想番地と実番地の対応関係が一意的に定義され，維持されていく必要がある．前節に述べたように，仮想番地はページ番号とオフセットに分割されており，仮想番地と実番地の対応は，ページ番号とページ枠番号の対応に置換えられる．ページ対ページ枠の対応関係は，ページング方式の趣旨からいって，アプリオリに特定の拘束的な条件を前提とすることは望ましくなく，したがって，その表現形式は対応するペアを列挙するテーブル形式によることになる．このテーブルは普通番地変換テーブルと呼ばれている．

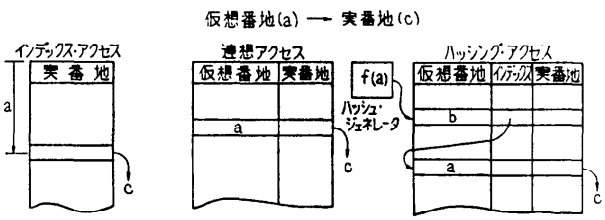

図-3　番地変換テーブルの構成方法

番地変換テーブルの構成方法には，つぎのようなものがある．図-3はこれらを図解したものである．

1)　インデックス・アクセス方式によるもの
2)　連想アクセス方式によるもの
3)　ハッシング・アクセス方式によるもの

1)の方式は必要なテーブル・エントリを探索する手間が不用である反面，テーブルの長さが仮想記憶空間の大きさに比例して膨れあがり，そのために過大な記憶容量を喰うことになる．この欠点は番地変換テーブル自身に特殊なページング方式を適用することによって，かなりの緩和を計ることができる．たとえば，IBM システム/370 の‘セグメント’は，一部この考え方に立つものとみられる．2)の方式は連想探索法を利用することによって，テーブル長の圧縮をもくろむものであるが，この場合のエントリ数はページ枠の個数に等しく，それだけの規模をもった高速の連想記憶を実現するのは極めて高価につく．この欠点を救う方法としては，経済的に許される規模で，連想探索機能を備えたバッファを設け，そこにテーブルの内容を一部づつ入れて探索を進めるというようなことが考えられている．3) の方式は 1) と 2) の方式を折ちゅうしたものとみることができる．この方式では，図-4に示したように二つのテーブルを用いる．ページ・ディレクトリは，本質的には連想テーブルとしての構造をもっており，各エントリはそれぞれ一つのページ枠に対応していて，ページ枠番号でインデックスされているもの

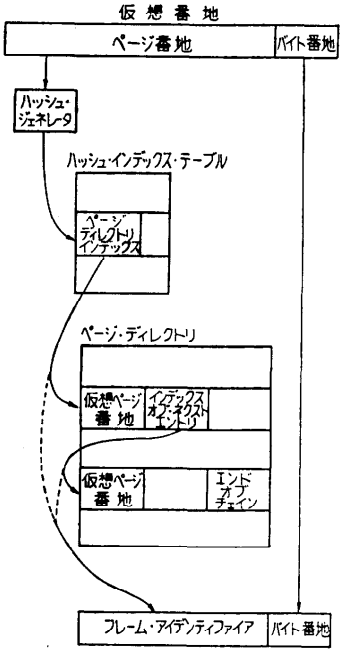

図-4　ハッシングを用いた番地変換テーブル

とみなすことができる．エントリの内容は一つのページ番号と，連結リストのためのインデックスその他の情報である．ページ番号はそのエントリに対応しているページ枠を占有しているページのものである．連結リストの起点は，もう一つのテーブルであるハッシュ・インデックス・テーブルの中に置かれている．いま，あるページ番号が変換のために与えられると，それが適当な方法でハッシュされて，ハッシュ・インデックス・テーブルのどれかのエントリを指定するインデックスに変換される．これによって選択されるエントリには，さらにページ・ディレクトリへのインデックスが入っているので，それをたどってページ・ディレクトリのあるエントリに至る．そこで，このエントリにあるページ番号と最初に与えられたページ番号とが比較され，一致すればこのエントリに対応するページ枠番号が番地変換として求めるべき番号であるということになる．もし，一致しなければ連結リストによって与えられるインデックスをたどり，ページ・ディレクトリの別のエントリについて，いまと同じことをする．これを繰返していると，ページ番号の一致が得られぬままに連結リストの終点に達してしまうことになる．これは与えられたページ番号に対応するページ枠番号が未定義であることを示している．

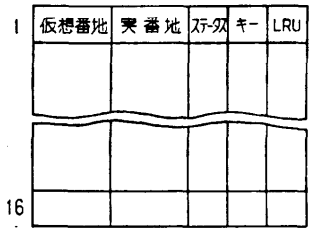

図-5　高速変換バッファ

番地変換は命令実行のたびごとに繰返され，余分な時間を喰うものであるから，できるだけ高速化する必要がある．このために考えられることは，番地変換テーブルを高速化してアクセス時間を小さくすることであるが，さらにそのうえの高速化を計るものとして高速変換バッファ（TLB とか連想記憶などと呼ばれることが多い）という工夫がある．これは，**図-5** に示したような表形式の構造をもった非常に高速な連想記憶で，通常は 16 エントリを 1 組とすることが多い．変換すべきページ番号が与えられると，全エントリが連想的に探索され該当するエントリがあれば，それから直ちに対応するページ枠番号が得られることになる．該当するものがなければ，本来の番地変換テーブルからそれを読出してくる．そして，バッファのエントリのうちで当分必要なさそうなものを選んで，いまのページ番号とページ枠番号の組を登録し，以後の参照に備える．エントリ中の制御ビットは必要なさそうなエントリを選ぶのに使用するもので，それによるバッファ管理の実際は，キャッシュ・メモリの場合に似ている．詳細は参考文献 1),3),6) などにゆずる．高速変換バッファが用いられるのは，プロセッサにおける番地変換のみで，入出力プロセッサではそこまでの高速性が要求されることはない．

以上の説明のまとめとして，ページング方式における番地変換の概要を**図-6** にあげておく．

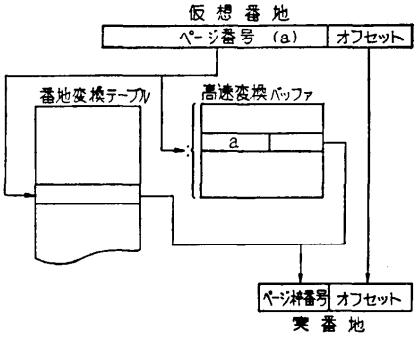

図-6　ページングにおける番地変換機構

（4） 記憶管理機構

記憶管理機構の役割は，システムに与えられている記憶管理の方針に従って，主記憶と二次記憶に対するページの配置と転送を実行することである．したがって，ここでの最大の問題は記憶管理方針のあり方ということになる．

主記憶に対する記憶管理方針は，前述したようにつぎの三つにブレークダウンされるが，

1) 置換えアルゴリズム
2) 配置アルゴリズム
3) 取込みアルゴリズム

多重プログラミング・システムの場合には，さらにつぎの二つの問題に対する方針を追加する必要がある．

4) 多重プログラミングの水準の制御
5) メモリ・パーティションの制御

4)は同時に主記憶に入ることのできるプロセスの個数をどう制限するかという問題であり，大別的には，この個数をいつも一定に抑える固定方式と，状況に応じて変えていく可変方式とがある．5) は 4) によって共存を許されることになったプロセスの間で，主記憶をどのように分け合うかの問題である．これにも原則的にいって固定方式と可変方式が考えられる．

ページングにおける主記憶管理方式の詳細については，本特集の別のところで紹介されているので[5]，ここでは省略して，つぎに二次記憶の管理方式について説明する．主記憶がページ単位に細分化されてページ枠が構成されたのと同じように，二次記憶も‘スロット’と呼ばれるページ単位の物理レコードに分割されていることは前述した．二次記憶管理の主要問題は，実行の対象となるプログラムの各ページに対して，いつ，どのようにスロットを割付け，いつ，どのようにページの転送を行うかということである．

スロットの割付け方式としては，前述したように固定方式と浮動方式がある．固定方式はプログラムの実行開始から終了まで，同じページが同じスロットに割付けられているものである．二次記憶として可動ヘッド・ディスクが用いられる場合には，同じプログラムに属するページはできるだけ同じか，あるいは隣接したシリンダに集中して置くというような工夫が行われる．浮動方式は，各ページに対してプログラムの進行とともに動的にスロットを割当てていくもので，プログラム実行中に内容が変更されたページをページ・アウトするときには，そのときヘッドが位置しているシリンダないしその近傍から新しいスロットを割当てて，ページ書出し時間を少なくするようにしている．

つぎに，ページ書出しをいつ，どのようにして行うかのページ転送方式については，転送要求の待ち行列に対して，それら全部を処理するために必要なシークと回転待ちの時間を，できるだけ少なくて済むようにするための，要求実行順序のスケジューリングとして，いくつかの方式が工夫されている．

4.2 セグメンテーション方式

（1） 原　理

セグメンテーション方式は，仮想空間の区分化をそこにロードされるプログラムの内部構造に即して行うものである．したがって，ページング方式の場合と異なり，この区分化はプログラマの指示にもとづいて行われ，論理構造のレベルにおけるサブプログラムや集団データなどといった論理的な実体を設定することと一対一に対応する操作になっている．このような論理的な実体と，それに対応している仮想空間のある区域を厳密に区別して扱う必要のある場合には，前者をオブジェクト，後者をセグメントと呼ぶ．特に厳密さを要求しない場合には，セグメントということばを両方の意味に用いるのが普通である．このような論理的実体の設定をサポートするためには，これらの実体を直接名指しできるような機構が必要である．このため，セグメンテーション方式では論理番地の構成を二段階にし，セグメントを指示するのに用いる番号――セグメント番号と，セグメントの中で命令やデータを指示するための番号――語（またはバイト）番号という二次元構造のアドレッシングを用いている．これは一見するとページングにおけるページ番号とオフセットという二段階の構造の同類として受取られやすいが，ページングの場合は論理番地はあくまでも一次元のままであることに注意する必要がある．いいかえれば，ページングによる仮想空間は一次元であるのに対して，セグメンテーションによるそれは，図-7 に示したように二次元空間になっている．

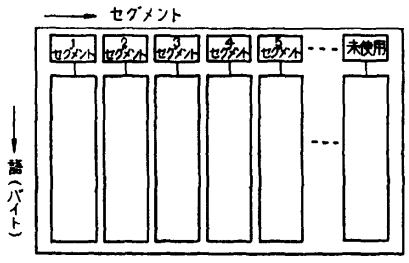

図-7　二次元仮想空間

セグメントは論理的実体としての性格上，その取扱いは多くの点でページの場合と異なってくる．仮想記憶の観点からみた最大の相違点は，セグメントの大きさが一様ではなく，場合によっては動的に伸縮を許す必要があるということである．これは記憶管理にさまざまな問題を持込むことになる．第2の相違点はセグメントに対しては仮想記憶としての制御と並行して，論理的実体としての制御，例えば情報の保護や共用のための制御が行われるので，考え方や機構が多元的になってくることである．したがって，それを純粋に仮想記憶だけの立場だけから議論するのは片手落ちになってしまう．これらについては後述する．

（2） セグメント・ディスクリプタ

セグメントはいろいろな観点からの制御を受けるものであるため，その管理を容易にする工夫として，こうした制御に関係するセグメントの属性をひとまとめにして記述する機構が用意されている．すなわち，セグメント・ディスクリプタである．図-8はMULTICSシステムで用いられているセグメント・ディスクリプタの例である．この例にも見られる通り，セグメント・ディスクリプタの内容は，仮想記憶管理に関係する属性を記述した基本ディスクリプタの部分と，情報保護・共用管理に関係するプロテクション・ディスクリプタの部分とに大別できる．基本ディスクリプタはセグメントの実番地と大きさを記述するものである．実番地は詳しくいうと，そのセグメントが主記憶上にあるかどうかの区別を示すビットと，主記憶上にある場合にはその実番地とで構成されている．プロテクション・ディスクリプタの内容はシステムによってさまざまであり，セグメンテーション方式の元祖ともいえるバロース B 5000 では，単にそのセグメントの中味が手続きであるかデータであるかといったタイプの記述程度であるのに対して，図に示した MULTICS の例では簡単に説明することが難しいほどの内容をもつようになっている．

いうまでもなく，セグメント・ディスクリプタはセグメントと一対一に対応するものである．しかし，手続きやデータの共用が行われる場合には，その解釈に若干の注意が必要である．すなわち，セグメントは仮想空間上の実在であるのに対して，共用が行われる場合に，それぞれの利用に個別のコピーを用意するかしないかは実記憶空間レベルでの問題であるに過ぎない．

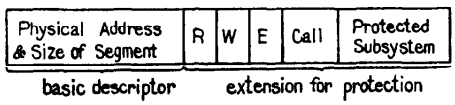

図-8 MULTICS セグメント・ディスクリプタ

（3） 番地変換機構

同じプロセスに属するセグメントは，それぞれ固有のセグメント番号が与えられ，それらに対応するセグメント・ディスクリプタは，そのセグメント番号でインデックスされるテーブルの形に配列されて取扱われる．このテーブルはセグメント・ディスクリプタ・テーブルと呼ばれ，番地変換表としての機能をも果たすものである．

セグメンテーション方式における論理番地は，二次元の形式で (x, y) として与えられる．ただし，x はセグメント番号，y は語またはバイト番号である．この x をインデックスとしてセグメント・ディスクリプタ・テーブルを引けば，x に対応するセグメント・ディスクリプタから，そのセグメントの実番地と大きさが得られる．y がセグメントの大きさを超えていれば，それはどこかに誤りがあることを意味する．超えていなければ，y にセグメントの実番地を加えたものが論理番地 (x, y) に対する実番地である．

高速バッファを利用してテーブル参照を高速化する工夫は，ページング方式の場合と同様ここでも有効である．

（4） 記憶管理機構

セグメントはそれぞれの大きさが自由に決められるため，記憶管理はページングの場合よりも複雑になる．記憶の断片化という新しい問題が発生してくるためである．プログラムの進行に従って，多数のセグメントが主記憶にロール・インされたり，ロール・アウトされていると，主記憶上には新しい記憶割付けに対して利用可能な領域が，何個所にも分かれて散在するようになる．そして，必要なセグメントをロードしようとしても，それらのどの個所にも大き過ぎて収容できないという現象が出現してくる．そうなると，これらの個所は隣接した部分が新しく解放されて領域が拡大されるのを待つか，あるいは積極的に主記憶上の内容を再配置するとかしない限り，使い道がなくなる．

主記憶の一部が空き地化して遊んでしまうという現象は，ページングの場合にも別の形で出現してくるものである．すなわち，プログラムをページに分割する場合，プログラムの大きさがページ長の整数倍になっていなければ，余りの部分を収容しているページには

空き地が含まれることになる．これを特に内部断片化と呼ぶことがある．その場合にはセグメンテーションにみられた上述の現象を外部断片化と呼んで区別する．

セグメンテーション方式における記憶管理方針の考え方は，ページングの場合と同じように参照の局所性を利用した主記憶利用の効率化とともに，記憶の断片化をいかに回避するかが重要な課題となってくる．これに関する具体的な提案については，文献 6) に詳しい説明がある．

4.3 多重化と複合化

（1） 多重ページング方式

プログラムが大規模化するのに伴って，それを収容する仮想空間の大きさも拡大され，論理番地の長さも 24 ビット，32 ビット，48 ビットなど増大の一途である．一方，仮想記憶を効率的に実現するのに適したページ長は，2K ないし 4K バイト近辺と考えられている．これからいえることは番地変換に用いられるページ・テーブルが非常に大きくなり，取扱いに不便と無駄を生ずるということである．また，手続きやデータの共用が行われる場合を考えると，例えば一つの手続き全体が完全に入り切るような記憶領域の単位があれば，番地割付けなどに好都合である．このようなことから，仮想空間をまず大きな単位で区分化したあと，それぞれをさらにページングすることが考えられる．すなわち，二段階のページングを行うわけである．IBM システム/370 などの例でみると，仮想空間はまず 64K ないし 1M バイトを単位として区分化され，それがさらに 2K ないし 4K バイトでページ化される．そして前者の単位区分を後者のページと区別するためにセグメントと呼んでいる．これはセグメンテーション方式でのセグメントと紛らわしいが，あくまでも一次元空間としての枠組の中に留まるものであることは，ページの場合と同様である．

このような2レベルのページングにおいては，番地変換テーブルもセグメント・テーブルとページ・テーブルの二段構造になる．番地変換の概要は**図-9** に示した通りである．変換が二段階になるため余分な時間を必要とすることになるが，高速連想バッファの利用によって，この欠点はかなりカバーされる．

2レベル・ページングの利点は，1) いくつかのプログラムの間で手続きやデータの共用を行う場合，それらをセグメントとして扱うことによって，ページ・テーブルも共用化することができ，ページの置換えなどにおけるテーブルの保守が容易になる．2) 使われていないセグメントに対しては，ページ・テーブルを用意する必要がなく，テーブルのためのスペースが節約できる．3) セグメントの大きさを適当に選べばページ・テーブル自身がページの中に収まることになり，ページ・テーブルに対する記憶管理が容易になる，などである．

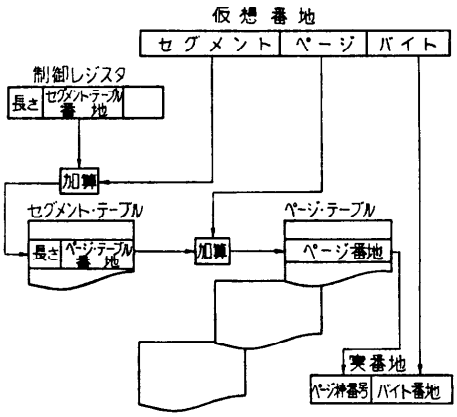

図-9 2レベル・ページングにおける番地変換

（2） セグメンテーション/ページング複合方式

セグメンテーション方式の狙いは，仮想記憶の実現と情報の保護・共用などのための論理的実体の設定という二つの課題を，一つの機構で同時に解決しようというところにあった．しかし，これにはセグメントの大きさに対する要求に，固定長：可変長という対立を含むところに本質的な難点があった．この難点を取除くためには，それぞれの課題を互に調和する別々の機構によって解決するという方策が考えられる．すなわち，仮想記憶の実現はページング方式により，論理的実体の設定はセグメンテーション方式によるという考え方である．

このとき，ページングとセグメンテーションの調和をどうとるかで，二つの行き方がある．一つは各セグメントごとにページ化を行うもので，MULTICS がその代表例である．この場合の番地変換機構は，二次元番地の取扱いを除けば，図-9 の2レベル・ページングの場合と形式的にはほとんど区別がつけられない．他の行き方は，まずページング方式によって，影の一次元仮想空間を展開し，そのうえにセグメンテーション方式による二次元空間を投影するという考え方をとるものである．この場合には，セグメントの境界がページの境界以外のところにおちても問題はない．ただ

この後者の行き方では，セグメントが動的に伸縮することがあると，影の一次元仮想空間上での再配置が必要になり，各セグメント・ディスクリプタを更新する手間がかかることになる．

このように，目的別に実現機構を分離することによって，それぞれの目的をより純粋に，徹底した形で追求することができるようになる．セグメントは，セキュリティ制御における保護対象を精密に区分するための基本単位として，あるいはサブプログラムやファイルを共用化するための機構として，さらには二次元アドレッシングのために，セグメントの内部構造を他のセグメントに影響を与えることなく（必要なリンケージは維持したまま），自由に書直すことができることから，モジュール化構造のための機構として，またいわゆるダイナミック・リンキングを可能にする機構として利用できるというように，多くの利点を有するものである．逆に，欠点としては2レベル・ページングと同様に，番地変換のためのオーバヘッドが大きくなってパフォーマンスを落とすことである．この欠点が連想記憶の利用によって緩和されることも同じである．

（3） 多重仮想空間

多重プログラミングの原始的なイメージは，複数のジョブが主記憶を分割的に共用するものであるが，仮想記憶方式を用いたシステムでは，これを実現するのに二つの形式が考えられる．第一のものは，並行処理されるジョブの間で単一の仮想空間を分割的に共用するものである．これは仮想記憶が導入される以前のシステムを，そのまま仮想記憶方式システムに取込むためにとられる形式であると考えられる．第二の形式は各ジョブごとに別々の仮想空間を与えるようにするものである．仮想空間を実在化する機構は番地変換テーブルである．したがって，仮想空間を複数化するには複数の番地変換テーブルを設け，それらをジョブごとに切換えて利用することができればよい．一般に番地変換テーブルの所在は，プロセッサにある制御レジスタに置かれたポインタによって指定されるようになっているので，テーブルの切換えは制御レジスタのポインタを入れ換えることで簡単に実現できる．

多重仮想空間方式の利点は，一つのジョブで使える番地空間が仮想空間いっぱいに拡大されること，したがって，あるジョブで別のジョブ内の番地を生成することが不可能になる（空間を異にすることになる）ため，論理的な意味での記憶保護が強化されることなどである．

5. 最近の動き

5.1 単一レベル記憶

‘単一レベル記憶’ということば，古くて新しいことばである．これが最初に用いられたのは，英国マンチェスタ大学のグループによってページング方式の考え方が導入されたときである．このときの‘単一レベル’という意味は，プログラム記憶として用いられる二次記憶を主記憶のレベルに等価変換して，主記憶と二次記憶のレベル差（明示的な入出力命令の存在）を解消するということであった．これに対して，最近の用例における‘単一レベル’の意味は，ファイル記憶としての二次記憶を主記憶のレベルに統合化すること，さらには多段階の記憶階層にまたがったデータベース記憶までを統一的な番地空間の中に取込んでしまおうとする考え方を表わすものとなっている．

ファイルをプログラムと同じ番地空間に取込んで管理するために必要なことは，まず，空間の大きさがプログラムと同時にいくつかのファイルを収容するに十分であり，さらに，ファイルが一つの論理的実体として直接アドレスできるようになっていることである．前節に述べたセグメンテーション/ページング方式がこの必要条件を完全に満足するものであることは明らかである．したがって，この方式の利点としてファイルの直接アドレッシングを追加すべきである．しかし，この新しい意味での単一レベル記憶を実現するためには，ただ単にこの方式を採用するというだけでは不十分で，さらに多数のオブジェクト（論理的実体）を体系的に管理するためのディレクトリ機構や，それらを記号的にアドレスするための機構が不可欠である．これらの問題については，文献 3) が詳しい．

多レベルのデータベース記憶までを取込むような単一レベル記憶については，まだ基礎的な提案が行われている段階で[7]，本格的な展開は今後の課題である．

5.2 仮想記憶管理のファームウェア化

オペレーティング・システムのファームウェア化はここ数年来の大勢的な傾向であり，これまでプログラムで実現されていた仮想記憶管理も，最近の新機種ではほとんどがファームウェア化されていることが多くなっている．これによって仮想記憶によるパフォーマンス上のオーバヘッドが一段と軽減されている．

6. あとがき

以上では仮想記憶を二次以上の記憶装置を主記憶レ

ベルに等価変換するものとみる立場に重点を置いて解説した．しかし，セグメンテーション/ページング複合方式の比重が増大しつつある現状からいえば，むしろ主記憶の概念をさらに抽象化して，そこに新しい論理構造を与えるものであるという観点をより強く打出すべきであったのかも知れない．そうしなかったのは現在の通念を離れて，ある特別な主張を行うことになるのを恐れたためである．

参 考 文 献

1) IBM: A Guide to the IBM 4341 Processor, GC 20-1877-0 (1979).
2) 西垣，緒方：仮想メモリ・システムの二次記憶管理方式の比較解析，情報処理学会論文誌，Vol. 20, No. 4, pp. 290-298 (1979).
3) Organick, E. I.: The Multics System, MIT Press (1972).
4) 中所，林：ページングアルゴリズムの性能に関する実験的および理論的解析，情報処理学会論文誌，Vol. 20, No. 6, pp. 460-467 (1979).
5) 益田：仮想記憶の制御方式，情報処理，Vol. 21, No. 4, pp. 369-377 (1980).
6) Denning, P. J.: Virtual memory, Computing Surveys, Vol. 2, No. 3, pp. 153-189 (1970).
7) Lam, C-Y, and Madnick, S. E.: Properties of Storage Hierarchy Systems with Multiple Page Size and Redundant Data, ACM TODS, Vol. 4, No. 3, pp. 345-367 (1979).

(昭和54年12月24日受付)