バーチャルメモリについて† I

吉広誠一†† 黒沢　格††

1. まえがき

現在，コンピュータの記憶装置は，高速度で大容量という要求を経済的に満たすために，何種類かのメモリが階層的に配置されている．その階層は，基本的には内部メモリと補助メモリの2つのレベルからなっている．この2つのメモリの違いは，単に前者が高速度，後者が大容量という性能の違いだけではなく，CPUが参照できるのは内部メモリにある情報に限る，という機能上の区別にもなっている．

このため，内部メモリ（主記憶）に入りきらない大きなプログラムの実行には，後述するようなオーバレイ処理をプログラムに書き入れなければならず，プログラマにとって大きな負担となる．

これに対しバーチャルメモリ方式では，内部メモリ（主記憶）と補助メモリを見かけ上一体のものとし，要求された情報が内部メモリにない場合には，補助メモリからの情報の転送がシステムにより自動的に行なわれる．プログラマは見かけ上，補助メモリが持つ記憶容量を提供され，オーバレイの煩わしさから解放される．

このような方式は十数年前に提案されており[1]，実用化された方式もありながら[2]，初期においては利用側の要求がオーバレイ処理の軽減，リスト処理言語にともなう問題の解決程度であったこともあって，その後の発展に時日を要している．近年になって計算機の利用拡大にともない，プログラミングの容易性，プログラムの互換性等の要求が強くなってきた．さらに，タイムシェアリングの発展にともないメモリを動的に管理する必要が増加し，バーチャルメモリの開発は一層促進され，現代のコンピュータ・システム技術の一つになった．

本稿ではこれを2部に分けて解説する．第I部ではバーチャルメモリの必要性とその機能を解説するために，通常システムで問題になるオーバレイおよび再配置について述べ，それと対比しながらバーチャルメモリ・システムの構成を明らかにする．第II部では，バーチャルメモリ・システムの効率について述べる．

用語の意味について

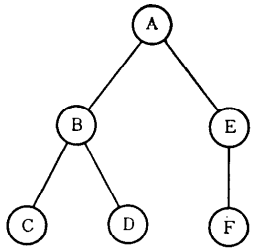
バーチャルメモリは“仮想記憶”と訳される．この“仮想”とはどういう意味であろうか．例えば，“バーチャル名人”という言葉があったとすると，ここでいう“バーチャル名人”とは，「名人たる証書は持たないが一般の人からは名人並みの実力者と扱われている」というような意味にとればよいであろう．すなわち，バーチャルメモリとはハードウェア的にはランダムアクセス・メモリということはできないが，プログラマからみれば補助メモリを使用してシュミレーションされた大容量ランダムアクセス・メモリとみてよいということで，ユーザは補助メモリが持つ大きなアドレス領域を用いてコーディングできる．このアドレス空間をバーチャルアドレス空間と呼ぶ．

2. オーバーレイ処理およびメモリ再配置とバーチャルメモリ

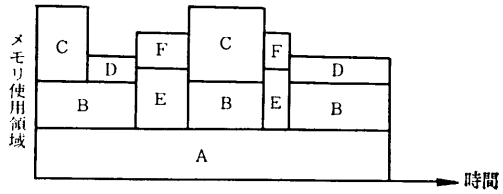
2.1 オーバーレイ処理

使用可能な内部メモリの容量にくらべプログラムが大きい場合には，これをいくつかの部分（セグメント）に分割して補助メモリに入れておき，プログラムの進行に応じて必要な部分を内部メモリへ呼び込むという技法（オーバーレイ処理）が必要になる．この方法では，どのセグメントをいつ，内部メモリのどこにロードするか（メモリ割り付け）が重要な問題となる．図1はオーバーレイ処理におけるプログラム・ツリーと，内部メモリに存在するセグメントの関係を示している．オーバーレイ処理では，まずプログラムをそれぞれ独立にロードされるいくつかのセグメントに分割する（図のA～F）．次に，プログラム・ツリーに従ってメモリ割り付けを決め，セグメントのロードに必要なI/O命令をプログラムに書き入れる．以上の手続きを，プログラマはプログラム実行以前にやらなけれ

† Virtual Memory
†† 電子技術総合研究所電子計算機部記憶システム研究室



（a）プログラム・ツリー



（b）メモリ使用領域の時間に対する変化

図 1　オーバーレイ処理

ばならない．これはかなり面倒な仕事であるばかりでなく，複雑なプログラムや複数プログラマによる共同作製の場合非常に煩雑になり，オーバーレイ処理に要するソフトウェア・コストの率は非常に大きくなる．

2.2　メモリ再配置

マルチプログラミングやタイムシェアリングでは，内部メモリに複数のプログラムがロードされる．その場合，個々のプログラムに対しては連続したメモリ領域を割当てなければならない．したがって，いくつかの空き領域があってもそれらの大きさが，ロードしたいプログラムより小さい時には使うことができない．（このようなメモリの使用効率の悪化をフラグメンテーション（断片化）という．）図2にその様子を示す．AとCのプログラムが終り空き領域が生じても，次に待機しているプログラムDがAまたはCより大きい場合にはロードできない．AとCを加えればロードできるとすれば，Bのプログラムの記憶位置を移動してA，C領域を結合すればよい．これをメモリ再配置という．このような操作を可能とするには，プログラムのメモリ内での番地が固定していてはならない．そのためにベースレジスタが用いられるようになった．

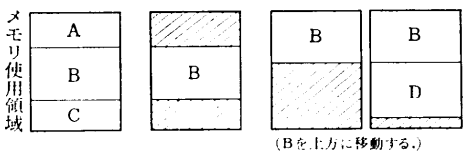


図 2　再配置

プログラムはあるメモリ番地（ベースアドレス）より始まるメモリ領域に相対アドレスでロードされる．そのベースアドレスがベースレジスタに入れられ，命令実行時にベースアドレスと相対アドレスが加算され内部メモリのアドレス（実アドレス）に変換される．あるプログラムから別のプログラムに実行が移る場合には，そのプログラムのベースアドレスがベースレジスタに入れられコントロールが切換えられる．

2.3　バーチャルメモリ[3),4),5),6)]

ところで，上に述べたオーバーレイ，再配置の仕事は本質的にみて次の4項目よりなると考えられる．

a）プログラムの分割．
b）セグメントのロード順序およびベースアドレスの明確化，
c）相対アドレスから実アドレスへの変換，
d）空き領域の結合．

バーチャルメモリでも同様にこれらの仕事に該当することを遂行しなければならない．バーチャルメモリでは内部メモリの大きさを意識することなく，プログラムの流れに従いコーディング可能であることは前にも述べた．コンパイルされた結果はまず機械的にまたはプログラムの論理的切れ目ごとに，適当な大きさのブロックに分割され（(a) に相当），補助メモリと内部メモリ間の情報転送の単位とされる．同時に，各ブロックの所在位置を示すテーブルが内部メモリ内に作られる．テーブルにはブロックが内部メモリに存在するか否か，および存在する場合にはその起点アドレス（ベースアドレス）が保持され，ブロックの移動とともにその内容が更新される．内部メモリに入りきらないブロックは補助メモリに記憶されている．図3にバーチャルメモリの基本的な機構を示す．バーチャルアドレスはブロック番号nとそのブロック内でのワード番号mの2つの数の組合せで表わされる．CPU より参照されたバーチャルアドレスは前記のテーブルを通して実アドレスに変換される（(c) に相当）．このテーブルをアドレス変換テーブルと呼ぶ．プログラムの進行につれて必要になった情報は，そのつどテーブルを通してアクセスされる．必要な情報が内部メモリにない場合には，その情報を含むブロックが補助メモリから内部メモリにロードされる（(b) に相当）．このようにしてオーバーレイはシステムにより自動的に処理される．

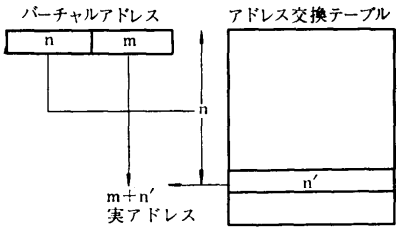


図 3.1　アドレス変換

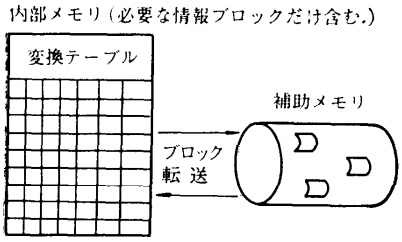


図 3.2　ブロック転送

図 3　バーチャルメモリシステム

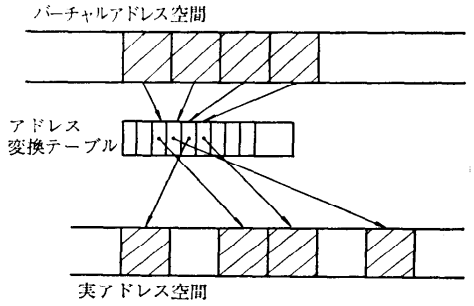


図 4　バーチャルアドレス：実アドレス対応システム

アドレス変換テーブルにより，プログラムの論理的な連続性が確保されるので，プログラムを連続したメモリ領域にロードする必要はなく，各ブロックは内部メモリのどこへでも割当てることができる．図4はその様子を示している．連続したバーチャルアドレス領域はブロックに分割され，内部メモリの不連続な領域に記憶される．したがって，複数個のプログラムを記憶する場合でも，各プログラムを任意の位置に分離してロードできる．（すなわち，再配置には **2.2** で述べたような方法と，このような方法があるということができる.）しかし，前述のフラグメンテーションは，小さくはなるが完全になくなるわけではない．

次に，以上のような特長を有するバーチャルメモリがどのように実現されるかを述べる．

3.　バーチャルメモリの実現法

バーチャルメモリの実現方式は，ブロックの種類により次の3つに分けられる．

① ページング方式：内部メモリが固定長のページに分割されていて，プログラムもそれに従って固定長ページに分割され，情報転送の単位とされる．

② セグメンテーション方式：プログラムは論理的な切れ目に従い可変長のセグメントに分割される．それに従いメモリも可変長のブロックに分けられる．

③ セグメンテーションページング方式：プログラムの可変長のセグメントが，さらに固定長のページに分割されてメモリに収められる．

以上いずれの方式をとってみてもその実現に重要なものは，アドレス変換機構と書き換え方式である．

a）アドレス変換機構：バーチャルアドレスを実アドレスに対応させる機構である．アドレス変換テーブルが中心で，変換を高速化するためにベースレジスタまたは連想レジスタが補助的に使われる．希望の情報が内部メモリにあるときは，この機構によりハードウェア的にアドレス変換が行なわれる．

b）書き換え方式：いつ，どこに必要な情報を書き込むかを決定する方式である．その内容は次の2つのアルゴリズムからなる．

① フェッチ・アルゴリズム：希望のブロックが内部メモリにない場合（ページフォルトまたはセグメントフォルト），そのブロックを補助メモリよりロードしなければならない．そのとき，いつ必要なブロックをロードするかを決めるのがフェッチアルゴリズムである．要求があってから割込み処理によってロードするデマンド方式と，必要となるブロックを予測に基づき持ってきておく方式とがあるが，デマンド方式が用いられる場合が多い．

② リプレースメント・アルゴリズム：必要なブロックを収めるのに十分な空き領域が内部メモリにない場合，いずれかのブロックを内部メモリから追い出さなければならない．そのブロックを選ぶアルゴリズムである．これには，下記のようないくつかの方法があるが実用機では LRU, WS の種々の変形が用いられている．FIFO，ランダムは実現が簡単なため実験機で用いられている．

- LRU (least-recently-used): 前回の参照からもっとも時間の経過しているブロックを追い出す.
- WS (working set): ワーキングセットに含まれないブロックを追い出す. ワーキングセットとは, 今より一定数前までの参照に現われたページの全体である.
- FIFO (first-in-first-out): 内部メモリにもっとも長く滞在しているブロックを追い出す.
- ランダム (random): ランダムにブロックを選んで追い出す.

ただし, ブロックが可変長のセグメンテーション方式では, セグメントの長さも書き換え方式の考慮の対象になってくる.

次に, システムの具体的な構成にふれておく.

3.1 セグメンテーション方式[7]

プログラムはコンパイル時に, 自動的にセグメンテーション (セグメントに分割) される. COBOL では SECTION 単位, FORTRAN では SUBROUTINE というようなものに該当する単位に論理的に分割される. 従ってセグメントの長さは一定ではない. セグメントの一つずつに対応してセグメント・ディスクリプタが作られ, その集合がディクショナリとして補助メモリ上にオブジェクトプログラムに連続して書かれる. ジョブの実行に先立って, これが内部メモリに呼び込まれセグメントテーブルを形成する. 図5にアドレス変換機構とセグメントディスクリプタを示す. bはセグメントの長さ, W′ は内部メモリ上でのセグメントの起点アドレス, Dは補助メモリのアドレスである.

セグメントポインタ・レジスタには, 各ユーザ・プログラムに対応したそのセグメントテーブルの起点アドレスが入れられる. バーチャル・アドレスはセグメント番号sと, そのセグメント内でのバイト変位Wからなり, セグメントテーブルを使って実アドレスに変換される. プレゼンス・ビットPが“1”の場合そのセグメントは内部メモリに存在するが, “0”の場合には内部メモリ上になく(セグメントフォルト), 割込みが発生し補助メモリ・アドレスDを利用して補助メモリから読み込まれる.

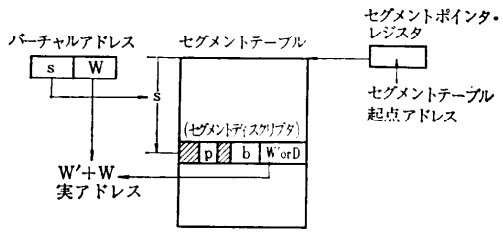


図 5　アドレス変換機構 (セグメンテーション方式)

内部メモリの空き領域は大きさ順にリンクされており, ロードされるセグメントにはその大きさに充分かつ最小の空き領域が割当てられる. 充分な大きさの空き領域がない場合には, 現在内部メモリにあるセグメントが追い出される. 内部メモリ内のセグメントも大きさ順にリンクされており, さらに, CPU からの参照時間も記録されている. それによって, 最近の一定時間内に参照のなかったセグメントのうちで, 最適サイズのものが選ばれ書き換えられる.

以上, アドレス変換の基本機構を述べたが, テーブル参照があるために処理速度が 1/2 程度に下がる. これを防ぐために専用のベースレジスタが用いられている. 連続した参照は同じセグメントに属する可能性が高い. そこである命令が実行された時, そのセグメントの起点アドレスをベースレジスタに入れておく. 次の命令ではそのセグメント番号と前のセグメント番号を比較し, 同じであればW+(ベースレジスタの内容)で実アドレスを得る. 異なる場合には上記のアルゴリズムによりセグメントの起点アドレスを得るとともに, ベースレジスタの内容も書き換えられる. このようにして高速変換が可能である.

3.2 ページング方式[4]

内部メモリは 2k (または4k) バイトのページに分割されている. バーチャル・アドレスは, セグメント番号s (8ビット), ページ番号p (5または4ビット), バイト変位W (11または12ビット) からなる. ここでいうセグメントとは, セグメンテーション方式におけるような論理的意味を持つブロックではなく, アドレス変換を2段階にして, テーブルに必要なメモリ領域を縮少したり, ユーザ共有部分の定義, メモリ保護などを容易にするためのものである. 図6にそのアドレス変換機構を示す. セグメントテーブル起点レジスタには, 各ユーザ・プログラムに対応したセグメ

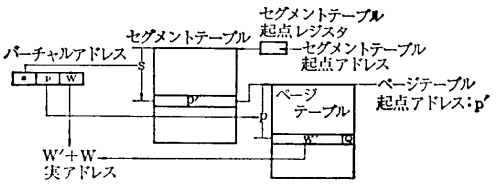


図 6　アドレス変換機構 (2レベル・ページ方式)

ントテーブルの起点アドレスが入れられる．セグメントテーブルのs番目には，そのセグメントに属するページテーブルの起点アドレス p′ が保持されている．次に p′ より始まるページテーブルの p番目から，そのページの起点アドレス W′ が読み出され，実アドレスW′+W が得られる．Qは要求されたページが内部メモリに存在するか否かを示すビットで，存在しない場合には(ページフォルト)，割込みが発生しスワップテーブルを利用して補助メモリ上の番地が知られ，内部メモリに読み込まれる．このとき内部メモリに空き領域がなければ，LRU アルゴリズムにより空きページが作られる．

高速変換を行なうのは，ここでは連想レジスタである．このレジスタには内部メモリに存在するページの内，参照される可能性の高い，いくつかのページの起点アドレスが記憶されており，バーチャル・アドレスの (s, p) 部分が参照されると高速度でその起点アドレスが読み出される．もし連想レジスタ内に存在しない場合は上記のアルゴリズムにより起点アドレスが得られる．それと同時に連想レジスタのエントリも最新の (s, p) および W′ を含むように，LRU アルゴリズムにより書き換えられる．

3.3 セグメンテーション・ページング方式[6]

セグメンテーション・ページング方式は，ページ機構を有するハードウェアを利用した場合のセグメンテーション方式ということができる．ページング方式では，バーチャル・アドレス空間は大容量の連続した空間が考えられたが，セグメンテーション方式ではセグメントに分れたものとして想定される．セグメンテーションページング方式ではセグメントの名前とそのセグメント内でのワード番号iとで語が指定される．セグメントの名前 (NAME) は KST (known segment table) を通して，システムによりセグメント番号sに変換される．セグメント番号sとワード番号iの意味は，セグメンテーション方式におけるセグメント番号sとバイト変位Wに全く同じである．ただ，そこで使われるメモリがページに分割されているために，ワード番号iの上位のビットはページ番号pに，下位のビットはバイト変位Wにわけられ，アドレス変換の機構は2レベルのページング方式同様2段階になる．その機構を図7に示す．セグメントテーブルのエントリ（セグメント・ディスクリプタ）にはそのセグメントに属するページテーブルの起点アドレス p′ とそのセグメントの長さb，セグメントへのアクセス権を示す

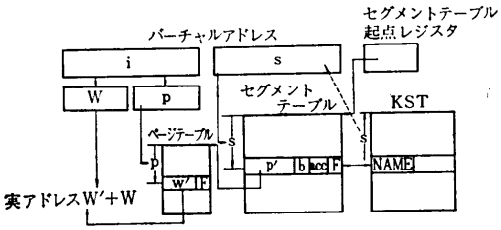


図7 アドレス変換機構（セグメンテーション・ページング方式）

acc，セグメントフォルトを示すビットFからなる．セグメントフォルト・ビットが“1”のときは，そのセグメントに属するページテーブルが内部メモリにできていないことを示している．

変換機構は見かけ上図6と同じであるが，2レベルページング方式とは異なり，セグメント番号sがページ番号，バイト変位とは全く独立に指定される．

情報の転送はページ単位で行なわれる．書き換えにはLRU アルゴリズムが使われている．連想レジスタを用いてアドレス変換を高速化している点も他の方式と同様である．

それぞれの方式の特長について簡単にまとめておく[3]．

共同利用を行なうバーチャルメモリシステムではプログラムの共有を容易に行えるのが大きな利点でありしかもセグメント単位でメモリ保護ができる．その他セグメンテーション方式では

1) プログラムが論理的な切れ目ごとに自動的にセグメンテーションされる．
2) 必要になったセグメントのみロードされるので，無駄を少なくすることができる．

次にページング方式では，

1) 内部メモリ全体に対し一様な取り扱いができる．すなわち，セグメントサイズを考慮することなしにプログラムをロードすることができる．
2) セグメントよりも一段小さい単位で情報の転送が行なわれるため，セグメント内での不使用部分はロードされない．そのために内部メモリの使用効率が上がる．

以上，バーチャルメモリの実現法とその特長を見てきたが，ハードウェア的には従来の記憶階層システムとほとんど変らない．（ただ，高速処理のための工夫がそれぞれ行なわれている．）したがって，図8に示すような拡張も可能である．この場合予備ディスクに

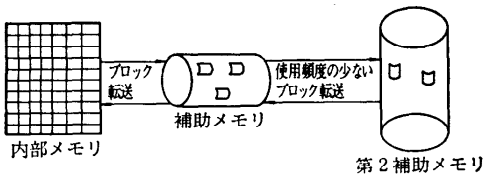


図 8 バーチャルメモリ記憶階層システム

は使用率の少ないブロックが記憶されることになる[4].

4. 第I部のおわりに

バーチャルメモリは通常中型機以上に採用される場合が多く，普通タイムシェアリング，リアルタイム処理，バッチ処理等を並列に走らせるようなオペレーティング・システムが用いられる．したがって，テーブルや OS に必要なレジデント・プログラムが大きくなり，ある程度以上大きい内部メモリが必要になる．

バーチャルメモリは，複数ユーザが各自大容量ランダムアクセス・メモリを持つかのように使用できるきわめてエレガントな記憶制御方式である．しかしこの方式の効率はどうであろうか．第II部ではその一端について述べる．

参 考 文 献

1) T. Kilburn, et al.: One-Level Storage System IRE Trans. EC., Vol. 11, No. 2, pp. 223～235 (1962).
2) Burroughs Corp.: The discripter——A definition of the B 5000 information processing system, Burroughs Corp. (1961).
3) P. J. Denning: Virtual Memory, Computing Surveys, Vol. 2, No. 3, pp. 153～189 (1970).
4) H. Katzan, Jr.: Storage Hierarchy Systems, Proc. SJCC, pp. 325～336 (1971).
5) 元岡 達編: 計算機システム技術, p. 41, オーム社, 東京 (1973).
6) A. Bensoussan, et al.: The Multics Virtual Memory: Concepts and Design, Comm. ACM, Vol. 15, No. 5, pp. 308～318 (1972).
7) 高千穂交易 (株): Burroughs B 6700/B 7700 System Conception, 高千穂交易(株) (1972).
8) E. I. Organick: Computer System Organization: The B 5700/B 6700 Series, p. 9, Academic Press (1973).

(昭和48年7月17日受付)