解　説

仮想記憶向きのプログラム作成方法†

小美濃　友夫††

1. はじめに

ここ数年来，ほとんどのOSで仮想記憶システムを採用してきている．ユーザサイドから見た仮想記憶システムを使用する上での利点として，

（1） 実記憶の大きさを意識せずに，大きなプログラムを作成する事が可能である．

（2） プログラムのオーバレイの事を気にする必要がない．システムが必要な時，自動的にオーバレイを行ってくれる．

等があげられる．

一方，逆に考えると上記の利点が仮想記憶システムを使用する上での欠点ともなりうる事がわかる．言い替えれば，仮想記憶システムでのプログラムの実行速度はプログラムの仮想記憶システム上でのふるまいによって影響を受けるので，プログラミングいかんによっては効率が悪くなる事があるという事である．この事は，ユーザにもある程度の仮想記憶システムに関しての知識が要求されている事を意味する．

本稿では，話を具体的にするために仮想記憶方式としてセグメンテーション方式を採用しているバロース B 6000/B 7000 システムを基に，仮想記憶システムのメモリ管理方式について概説し仮想記憶向きのプログラミング方法について具体例を用いて考察する．

2. 主記憶の割当てと管理

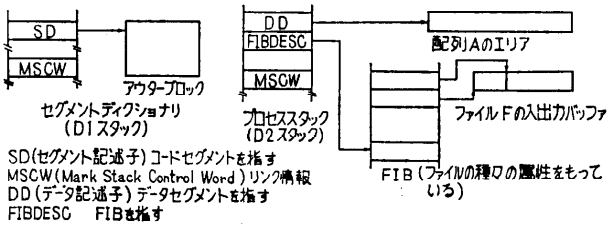
本章では，B 6000/B 7000 システムでのメモリの割当てとその管理方法について概説する．

2.1 メモリ割当ての単位

B 6000/B 7000 システムにおけるメモリ割当ては，セグメントと呼ばれる可変長サイズの単位で行われる．このセグメントは，各コンパイラによってその言語に適した単位で決定される．例えば，ALGOLに対しては，ブロック，ARRAY単位に，FORTRANに対しては，サブルーチン，DIMENSION 単位に，COBOL に対しては，セクション，01 レベルのレコード単位等である．

```
begin
  file F(kind=printer);
  array A[0:9];
  integer I;
    ⋮
  write (F,/,for I:=0 step 1 until 9 do A[I]);
    ⋮
end.
```

（a）



（b）

図-1 簡単なプログラムとそのメモリ割当ての概念図

これらのコードセグメントやデータセグメントは，プログラムの論理的な単位と考えられ仮想記憶制御の対象となる．これらのセグメントがメモリ上に割当てられた時の様子を示したものが図-1である．図-1からわかる様に，コードセグメントはセグメント記述子によって，またデータセグメントはデータ記述子によって管理されている．

2.2 メモリ管理とオーバレイ

メモリは，次に示す様な性質を持ついろいろなエリアに分割されて管理されている．

- 使用中エリア
  - オーバレイアブルエリア
    - リードオンリエリア
    - リード/ライトエリア
  - セーブエリア
- 未使用エリア

セーブエリアとは，プロセスが生存している間メモリ上に常に確保されているエリアで，スタック，FIB（ファイルのいろいろな情報を持っている），入出力バッファ，およびドープベクタがこれに相当する．オー

† The Programming Methods in Virtural Memory System Enviroment by Tomoo KOMINO (Online-system Development Division, Japan Information Processing Service Co., Ltd.).

†† 日本電子計算(株)オンライン開発室

バレイアブルエリアとは，プロセスが生存している間オーバレイの対象となるエリアで，これにはリードオンリなエリアとリード/ライト可能なエリアとがある．リードオンリエリアとしては主としてコードセグメントが，リード/ライトエリアとしては通常のデータセグメントが相当する（図-1 参照）．

メモリのオーバレイの方法としては次の様な方法がある．

（a） メモリからメモリへのオーバレイ

メモリエリアをオーバレイしようとした時，他に充分な未使用エリアがあればそのエリアへ移す．

（b） メモリ消去によるオーバレイ

（a）によるオーバレイが不可能の時，オーバレイしようとするエリアがリードオンリであればそのエリアをそのまま解放する．このエリアはコードファイル上にあるので，必要な時コードファイルから読むことができる．

（c） メモリからディスクへのオーバレイ

上記のオーバレイの対象にならないエリアは，そのエリアをディスクへ書き出すことによってオーバレイする．

2.3 メモリスワッピング

TSS ジョブの様なジョブは，トランザクションが発生するたびに処理すればよいので，これらのプログラムは常にメモリ上に存在する必要はない．

メモリスワッピングとは，処理すべきトランザクションがなくなったり割当てられたスライスタイムを費やしてしまったプログラムをディスクへスワップアウトし，以前にスワップアウトされていたプログラムをスワップインして処理をしようとするものである．このメモリスワッピングは，スワップエリア（あるいはサブスペースとも言う）と呼ばれる一つの連続したメモリエリアを使用する．スワップエリアはシステム全体のメモリ空間から見ればセーブエリアとして確保される．

各スワップジョブ（スワップエリアで走るジョブ）に割当てられたスワップエリア上のメモリエリアは，そのプログラム自身のメモリリンクによって管理されるが，メモリの割当ては連続した自分自身に割当てられたエリア内で行われるので，他のプログラムのメモリエリアをオーバレイする事はない．要求されたメモリが確保できないときは，そのジョブはスワップアウトされ，次にスワップインする時にメモリエリアが拡張される．

3. 仮想記憶システムにおける使用上の問題点とプログラミング上の留意点

多重処理環境で走るプログラムまたは実記憶に対して仮想記憶が非常に大きなプログラムを作成する場合には，本章の各節に述べている様な点に注意すれば効率の良いプログラムを作成する事が可能である．これらは相反する事柄であり，度を越すとかえって効率の低下を招くが適当に使用すれば有用である．適度の規準は使用している主記憶のサイズにもよるがここでは当社での経験（主記憶 400 K 語，スワップエリア 20 K 語，バッチの多重度 4 の時の）を基に述べる．（他メーカの計算機の仮想記憶システムにおいては，以下に述べる以外の留意点もあると思われる．）

3.1 セグメンテーション

コードセグメントのサイズは，各ブロック単位で決定され通常は分割される事はない．しかし，データセグメントの場合は分割される事がある．ALGOL で言えば，配列行（一次元配列の場合マザーベクタの指しているエリア，多次元配列の場合最後のドープベクタの指しているエリア）の大きさが 1 K 語を越えた場合にその配列行は 256 語ごとに分割される（但し，1 K 語を越えない時はその大きさがとられる）．FORTRAN の場合は，一次元，多次元を問わず一次元に写像されているため ALGOL の一次元配列の場合と取扱いが同じであるが，配列の大きさが 4 K 語を越えた場合には 256 語ごとに分割される（図-2）．

3.2 メモリの断片化

セグメンテーション方式の場合のメモリの断片化の原因は，いろいろなサイズのセグメントが混在する事による．極端に大きなエリアを定義したり極端に小さなエリアを定義したりすることが，ジョブ多重度を増した時にプログラムの効率を落しシステム全体のバランスをくずす原因となる．

次の様に定義された配列を考えて見よう．

array A[0 : 19], B[0 : 39], C[0 : 19]; （a）

これらの配列が使用されるとメモリ上には小さなエリア（（a）の場合 1 エリアあたり 4 語のメモリリンク語

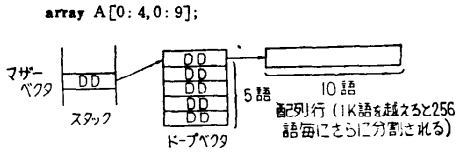


図-2 二次元配列のメモリ割当ての例

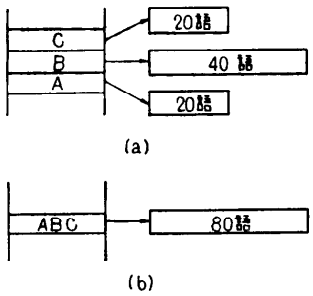


図-3 複数の配列の統合

を含めて24語，44語，24語）ができてしまう．これを次の様に定義したとする．

array ABC[0 : 79]; (b)

配列 ABC の最初の 20 語を A，次の 40 語を B，最後の 20 語を C として使用するわけであるが，この様な定義をすればメモリ上はある程度の大きさ（(b)の場合メモリリンク語を含め 84 語）のエリアを一つ確保するだけで良い(図-3)．同様に，10語の配列を100個定義すると合計1,400語必要であるが100語の配列を 10 個定義すると合計1,040語のエリアで良い．

但し，プログラム上での論理の展開がいくらか複雑となるので，理解しやすいプログラムを書くという立場からは，いくつかの配列を一つにして部分的に使用するというプログラミング技法は好ましくない．

3.3 スラッシング

スラッシングとは過度のオーバレイのため，メモリと二次記憶とのデータの授受が頻繁になりプログラムがほとんど止まった状態になることをいう．

システムがスラッシング状態になる主たる要因は，バッチの場合，

（1） ジョブ多重度が多くなる事によって未使用エリアが少なくなる．

（2） ジョブ多重度がそれ程多くなくとも大型のジョブが走ったため未使用エリアが少なくなる．

等であるが，TSS ジョブの場合は，すでに述べた様にスワップエリアと呼ばれる小さなエリア内でメモリの確保が行われるためにスラッシング状態になりやすい．

TSS ジョブでは，通常コードセグメント，データセグメントともスワップエリア内に割当てられる（データセグメントのみスワップエリアに割当てる事も可能である）．この事は，実行中のある時点を考えた時必要なコードセグメントとデータセグメントが同時に存在しなければならない事を意味する．この場合，m スロット（スロットとはスワップエリアを割当てる単位で1スロットが990語に相当する）のエリアが確保されており，n 個のデータセグメントが実行に際して必要であると仮定した時，n 個のデータセグメントすべてが同時に割当てられなければオーバレイが行われる．割当てられた m スロット内でオーバレイできない時はそのジョブはスワップアウトされ（この時セーブエリア，オーバレイアブルエリア全体がスワップアウトされる），次のスワップイン時に $m+1$ スロットのエリアに拡張される．mスロット内でオーバレイが行われる場合，必要な n 個のデータセグメント間で過度にオーバレイがなされた時に，スラッシング状態となりうる．

スラッシング状態になると，主記憶とディスクとの間にデータの授受のために入出力チャネルの使用率が高まり，スラッシングを起しているプログラムの効率さらにはシステム全体の効率を落す事になる．

特に TSS ジョブでスラッシング状態になるのを避けるためには，ユーザとしてプログラミング上，次の様な注意が必要である．

（1） コードセグメント，データセグメントの大きさは，スワップエリアの大きさに見合った適度な大きさにする．

（2） セーブエリアの大きさに注意する．（特にファイルのバッファの大きさと数に注意する．）

ユーザの立場としては，システムがスラッシング状態を検出し，システムが適切な処理をするのが望ましい．

3.4 ワーキングセットとプログラムの局所性

図-4 の断片プログラムは二次元配列のアクセス方法を示したものである．(a)の方法は，二次元配列Aを行列の順に参照している．列方向は配列行なので，

```
array A[0 : 10, 0 : 100];
for i:=0 step 1 until 10 do
  for j:=0 step 1 until 100 do
    begin
      A[i,j]:=…;
    end;
```

(a)

```
array A[0 : 10, 0 : 100];
for j:=0 step 1 until 100 do
  for i:=0 step 1 until 10 do
    begin
      A[i,j]:=…;
    end;
```

(b)

図-4 二次元配列のアクセス方法

添字 i に対して添字 j の変化する間は同じエリアを参照する．添字 i が変わった時に他のエリアを参照する様になるが，すでに参照されたデータセグメントは再度参照される事はない．

一方，(b)の方法は，同じ配列Aを列行の順に参照しているので，添字 j に対し添字 i が変わるたびに異なるデータセグメントが参照され，さらに添字 j が変わると，すでに参照されたセグメントが再度参照されてしまう事になる．

この例では，(b)よりも(a)の方がデータの局所性が高いと言えワーキングセットの大きさも小さくなる．ちなみに，筆者の知っている例では，ジョブ多重度1の時(b)の型式で書かれたプログラムで経過時間1時間であったのが，(a)の型式にした場合，同一条件で経過時間 12 分で終了した例がある．

次の例（図-5）は，局所的な配列を定義している手続きをループの中で呼び出される例である．この例では，手続きが呼び出されるたびに局所的な配列が割当てられたり解放されたりする．この局所的に定義された配列を大域的に定義すれば，手続きを呼び出すたびに配列が割当てられたり解放されたりしないので，オーバヘッドの少ない効率の良いプログラムとなる．

しかし，ユーザにとっては(a)の方が(b)よりもより理解しやすいプログラムなので，ユーザとしては，(a)の様に書いてもシステムが(b)の様に処理してくれるのが望ましい．

三番目の例は，不必要に大きなコードセグメントを

```
begin                          begin
  boolean B;                     boolean B;
  procedure P;                   array A[0:100];
  begin                          procedure P;
    integer I;                   begin
    array A[0:100];                integer I;
    :                              :
    A[I]=…;                        A[I]=…;
    :                              :
  end P;                         end P;
  :                              :
while B do                     while B do
  begin                          begin
    :                              :
    P;                             P;
    :                              :
  end;                           end;
end.                           end.
```

(a)　　(b)

図-5　ループ中で呼び出される手続きの例

```
if i=1 then                    procedure PA;
  begin  }                       begin  }
  :      } A                     :      } A
  end    }                       end PA; }
else if i=2 then               procedure PB;
  begin  }                       begin  }
  :      } B                     :      } B
  end    }                       end PB; }
else if i=3 then               procedure PC;
  begin  }                       begin  }
  :      } C                     :      } C
  end;   }                       end PC; }
                                 :
                               if i=1 then PA
                               else if i=2 then PB
                               else if i=3 then PC;
```

(a)　　(b)

図-6　コードセグメントの分割化

持つプログラムの例である（図-6）．この例では，A, B, C のそれぞれのコード部分は同時には実行されない．それにもかかわらず，コードセグメントとしては同一のセグメントとなっており全体として大きなエリアとなっている．これら A, B, C の部分をそれぞれ独立した手続きにすれば，異なるコードセグメントとなるので小さなメモリサイズで実行する事が可能である．この様にコードセグメントを分けたために，オーバヘッドが 80% から 20% に減少した例がある．

4. むすび

仮想記憶を効率良く利用する方法について具体例を用いて述べてきたが，これらの諸方法は必ずしも両立するものでなく相矛盾するものも多々ある事は言うまでもない．しかし，述べてきた方法を用いる事によってより効率良く仮想記憶システムを使用する事が可能となる．

最後に，有益なご意見をいただいたオンライン開発室各位に深謝する．

参考文献

1) Organick, E. I.: Computer System Organization, Academic Press, New York (1973).
2) Burroughs: B 7000/B 6000 ALGOL Reference Manual.
3) Burroughs: B 7000/B 6000 FORTRAN Reference Manual.
4) Burroughs: B 7000/B 6000 System Software Operational Guide.

（昭和 54 年 12 月 4 日受付）