2 次元番地付方式による HITAC 5020 TSS の特徴*

本林　繁**　益田　隆司**　勝枝　嶺雄**　高橋　延匡**

Abstract

In consideration of recent rapid progress in information processing, we have implemented a time-sharing system with two-dimensional addressing feature.

The HITAC 5020 TSS is organized on the computer system which has some additional hardware features to the conventional HITAC 5020 system.

The time-sharing system of this type has lots of merits in comparison with the conventional time-sharing system and will be the basis of future large scale information processing system.

In this paper, we first describe the segmentation and paging mechanism, and then discuss some parts of the supervisor which are characteristic of two-dimensional addressing, especially swapping of programs, and how to process common segments.

1. まえがき

将来の情報産業の方向は，好むと好まざるとにかかわらず，集中的なシステム化が促進されると予測される．その代表的なものの一つに，大型機によるタイムシェアリング・システムがある．MIT のプロジェクト MAC は，1965 年に CTSS の経験を生かし，かつ，将来の情報産業の特色である情報のサービスを盛り込んだ MULTICS システムを発表した．そこでは新たに，ハードウエアに対して，セグメントという概念を導入した．これによって，プログラムの存在領域が，セグメントとロケーションという 2 次元の領域に拡張された．

HITAC 5020 タイムシェアリング・システム（以後 5020 TSS と略す）は，このような 2 次元番地付方式を実験的にインプリメントし，それによって

（1） 2 次元番地付方式の TSS の設計方式を（ハードウェア，ソフトウェアを含めて）確立する．

（2） ファイル・システムの方式を確立する[4]．

（3） 人間とコンピュータとの対話の本質を解明する．

（4） ソフトウェアの生産性向上の研究を促進させる．という効果を得るべく開発した．

本報告では，**2.** でシステム構成およびハードウェア・メカニズムを，**3.** でスーパバイザのうちで，特に，2 次元番地付方式に付随する特徴的な点を取り上げて述べる．

2. 5020 TSS の概要

2.1 システム構成

5020 TSS のシステム構成は，**Fig. 1** で示すとおりである．ここで，TRC (Transmission and Reception Controller) は，通信回線で結ばれた多数端末装置との情報の送受信を制御する計算機で，5020 cpu と主メモリを共有している．両計算機は互いに相手に割込

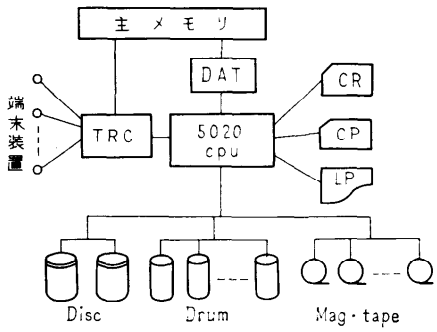

Fig. 1 Hardware Configuration

* The Characteristics of HITAC 5020 TSS with Two-Dimensional Addressing Feature, by Shigerlu Motohashi, Takashi Masuda, Mineo Katsueda and Nobumasa Takahashi (Centrol Research Laboratory, Hitachi, Ltd.)

** (株) 日立製作所中央研究所

みをかけることによって交信することができる.

5020 cpu にはチャネルを介して，種々の入出力装置が接続され，そのうちディスクは，ファイル格納用として使用し，ドラムはスワップ用に使用している．さらに，5020 cpu には，5020 TSS で重要な役割を果たし，2次元番地付方式を実現するためのハードウェア DAT (Dynamic Address Translator) が付加されている.

2.2 ハードウェア・メカニズム

5020 TSS 用に新規追加された TRC および DAT について，その機能の概要を述べる.

(1) TRC[8]

TRC は端末のユーザと，中央の計算機との入出力情報の送受信を制御するものであり，現在最大 32 回線まで接続できるように設計されている．TRC は次のようないくつかの特徴を有している.

(a) 多数入出力回線からの入力，出力を並列制御するハードウェアを有する.

(b) TRC は 5020 cpu と主メモリを共有するプログラム内蔵方式の計算機で，命令系は 5020 cpu のサブセットで構成されている.

(c) TRC と 5020 cpu とは，互いに相手に割込みをかけることができ，両者間の情報のやりとりは共有メモリを介して，割込みの制御によって行なわれる.

(2) DAT

DAT は 5020 TSS において，2次元番地付方式を実現するために，付加されたハードウェアである．2次元番地付方式では，プログラム内の各語を指定するのに，セグメント名およびセグメント内のロケーションからなる2次元的な論理アドレスが用いられ，実際にメモリへのアクセスが行なわれるときに，この論理アドレスが DAT により，MT (Mapping Table) を介して，主メモリ内の絶対アドレスに変換される．このアドレス変換のために使用される MT は，スーパバイザによって動的に作成，管理される.

DAT は，さらにページングの機能を有し，プログラムおよびデータは，ページと呼ばれる一定の長さの単位に分割され，メモリの管理を容易にするとともに，ページ単位のリロケーションが可能となる．MT 自身もページングされている．5020 TSS では，システム全体の複雑さを避けるために，ページの大きさを 256 語に統一した.

MT は SPDT (Segment Page Descriptor Table), SDT (Segment Descriptor Table), LPDT (Location Page Descriptor Table) の3段階のテーブルからなる.

SDT は各セグメントを定義づけるセグメント・デスクリプタの集まりであり，LPDT は個々のセグメントの各ページを定義づけるページ・デスクリプタの集まりである．セグメント・デスクリプタはそのセグメントに対応する LPDT の先頭アドレスをさし，ページ・デスクリプタは対応するページの先頭アドレスをさしている．SPDT は SDT をページングしたとき，その各ページの先頭アドレスをさすデスクリプタの集まりである.

以上のような MT によって構成される一つのアドレス空間は，SPDT の先頭アドレスをさす DBR (Descriptor Base Register) によって一意的に決定される (**Fig. 2**)．そして，この DBR の内容を入れ換えることによって，プロセスのスイッチが行なわれる.

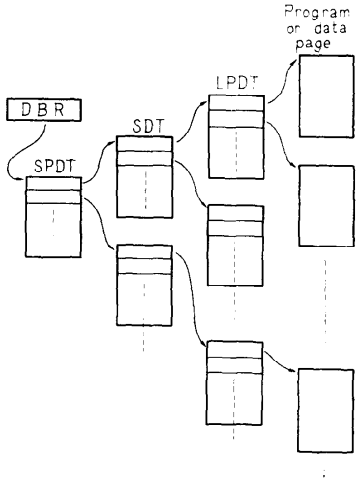

Fig. 2 Address Space Formed by Mapping Tables

MT を介して，2次元的な論理アドレスが，主メモリ内の絶対アドレスに変換される過程を図示すると **Fig. 3** のようになる.

以上述べたような，MT によるアドレス変換をメモリ参照のたびに繰り返すむだを避けるために，セグメント内のあるページが参照されると，セグメント番号およびページ番号（すなわち，論理アドレスの上位 24 ビット）と，そのページの先頭アドレス（すなわち，ページ・デスクリプタの内容）との対応表がアソシア

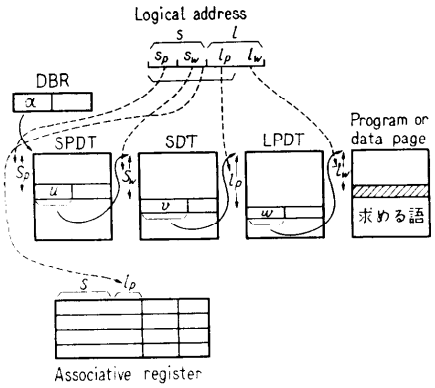

Fig. 3 Address Translation Mechanism of DAT

ティブ・レジスタに入れられ，同じページに対するメモリ参照の間は，MT を介さずに，直ちに絶対アドレスが求められるようになっている．

プログラム実行中に，命令の読み出しとかデータの参照などの際に，メモリ参照のもととなる論理アドレスを得るために，セグメント番号を記憶している次の3種のベース・レジスタが使われる．

（a） PBR—現在実行中のプログラムのセグメント番号を記憶する．

（b） ABR—データとしてメモリ参照を行なう際，対象となるセグメント番号を記憶する．

（c） JBR—ジャンプ命令のジャンプ先のセグメント番号を記憶する．

命令読み出しの際の論理アドレスは，PBR の内容と，現在実行中のセグメント内の，ロケーションを制御している逐次制御カウンタの内容を組み合わせて得られる．命令によってデータを参照する際の論理アドレスは，ABR の内容と命令内のアドレス部を組み合わせることによって得られ，さらに，ジャンプ命令のジャンプ先の論理アドレスは，JBR の内容とジャンプ命令のアドレス部を組み合わせることによって得られる．

（3） メモリ・プロテクション

5020 TSS のプロテクションメカニズムは，いわゆる，リング・プロテクションと呼ばれているもので，従来のように，コア・メモリ上での範囲で規定するものではなく，各ロジカルなセグメントごとにプロテクションが可能である．プロテクション情報は，各セグメントのセグメント・デスクリプタ内に，2ビットのアクセス・コントロールおよび4ビットの lock があり，各プロセスのアドレス空間を規定する DBR 内に4ビットの key があり，これらの組み合わせによって，プロテクションを行なう．

アクセス・コントロールは，個々のセグメントが有する性質を記述するもので，2ビットの組み合わせにより，read-execute 可能，read-write 可能，すべて可能，read only の4種類を表わす．一方，key と lock は各々のセグメントを，その重要度に応じてグループ分けしたときに，参照しようとするセグメントと，参照されるセグセメントが属するグループの相対的な重要度に応じて，プロテクションをかけるものである．

アクセス・コントロールおよび key と lock の組み合わせによるプロテクション・メカニズムをまとめると，**Fig. 4** のようになる．

アクセス コントロール	Key = 0			Key < Lock			Key = Lock			Key > Lock		
	R	W	E	R	W	E	R	W	E	R	W	E
0 0	○	○	○	○	○	×	○	○	○	○	×	×
0 1	○	○	○	○	×	×	○	×	○	○	×	×
1 0	○	○	○	○	○	×	○	○	×	○	×	×
1 1	○	○	○	○	×	×	○	×	×	○	×	×

R……Read，　W……Write，　E……Execute

Fig. 4 Protection Mechanism of DAT

Fig. 4 で○はアクセスが許されるが，×の場合のアクセスに対しては割込みが生じ，制御はスーパバイザに移行する．Fig. 4 からわかるように，key=0 のときは lock，アクセス・コントロールのいかんにかかわらず，すべてのアクセスが許される．これはスーパバイザのうちでも，中心となる部分にのみ適用している．

key≦lock のときには，アクセス・コントロールに見合ったアクセスが許され，key>lock のときは read だけが許され，write は決して許されない．key と異なる lock を有するセグメントへ飛び越そうとすると，必ず割込みが発生する．

3. スーパバイザ（特に，セグメント・メカニズムとの関係）

3.1 主メモリと2次メモリ間のスワップ

2. で述べたような，MT を介するアドレス変換の過程で，参照しようとしたページが，主メモリになかった場合には，ミッシング・ページ・フォールトと称する割込みが生ずる．この割込みは各デスクリプタ内の特定のビットが，オンになっている場合に生ずるものである．

5020 TSS では，あるページが主メモリにない場合には，対応するデスクリプタ内に，そのページの2次メモリ上のページ・アドレスをセットしておく（**Fig. 5**）.

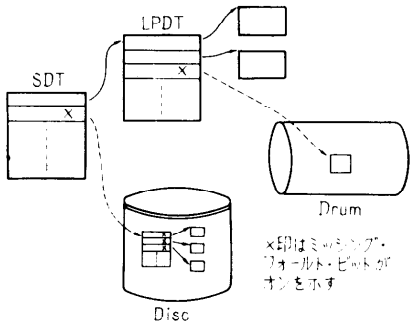

Fig. 5 Swapping between Main Memory and Secondary Memory

これはスワップの際の情報の節約という意味と，スワップの処理を統一的にするためである．すなわち，主メモリのあきがなくなり，ページをスワップ・アウトしなければならないときには，そのページをドラムのあきページに退避して，対応するページ・デスクリプタに “page not in core” のビットを立て，同時に退避したドラム上のページ・アドレスをセットする．その後，そのページが参照されて，ミッシング・ページ・フォールトの割込みが生じたときには，そのデスクリプタ内のドラム上のページ・アドレスから，主メモリのあきページにスワップ・インされる．

いま述べたのは，主メモリとドラムとのスワップであるが，ページが主メモリにないとき，デスクリプタ内にディスク上のページ・アドレスがセットされることもある．これはプログラムあるいはデータを，ディスクから最初にロードする場合に適用される．すなわち，ディスク上には，各々のファイルごとに，ファイル・ページ・テーブルと呼ばれる各ページに対する，ディスク上のアドレスを記述するテーブルが作られており，セグメントが割り当てられて，新しくセグメント・デスクリプタが登録されるときに，同時に，このファイル・ページ・テーブルのディスク上のアドレスがセットされる．そして，このセグメントが，実際に参照されると，まず，セグメント・デスクリプタでミッシング・ページ・フォールトがおこり，ファイル・ページ・テーブルが LPDT としてロードされる．さらに，もう一度参照されると，所望のページがディスクからロードされる．このように，ファイルをロードする場合にも，スワップの処理として，統一的に扱かうことができる．

以上述べたようなページング，およびミッシング・ページ・フォールトの機能により，次のような利点がある．

（1） ページ単位のリロケーションが可能である．

（2） 主メモリの容量制限を気にすることなく，大きなプログラムを作成することができる．

（3） 主メモリには，必要なページだけがあればよいし，実際に参照されたページだけがロードされるので，主メモリを有効に利用でき，不必要に大部分をスワップ・アウトするというむだも避けられる．

（4） 主メモリおよび2次メモリは，ページ単位に管理を行なえばよく，メモリの割付け，管理が，非常に単純化される．

3.2 リエントラント・プログラム

多数のユーザ間で，マルチ・プログラミングを行なうときには，主メモリを有効に使用するために，あるユーザが，すでに使用していたプログラムを，他のユーザがコールした場合，むだなコピーをせずに，同一のものが共有できることが望ましい．このようなプログラムは，リエントラント形式で書かれていなければならず，プログラムはその実行中に変化しない純粋プロシージャ部と変化を受ける作業用エリア部とに分けられ，作業用エリア部はユーザごとに別々に持ち，純粋プロシージャ部は共通に使用する．セグメントの概念を利用すると，このようなリエントラント・プログラムに対する扱いが非常に簡単になり，**Fig. 6** に示

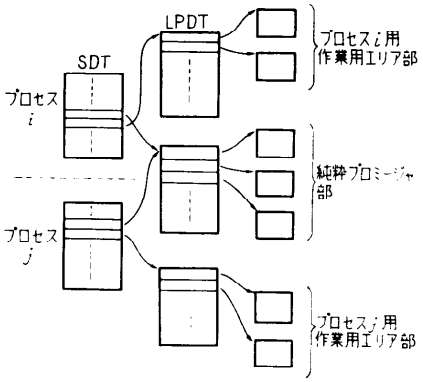

Fig. 6 Relation between Pure Procedure and its Working Area

すように，純粋プロシージャ部はユーザごとに割り当てられるセグメント番号が異なっていても，一つの共通の LPDT に結びつけられることによって共有される．一方，作業用エリア部は，各々のアドレス空間内に，別々のセグメントとして割り当てられる．

実際にリエントラント・プログラムが実行されるときには，それ以前のコーリング・シークエンスにおいて，先に述べたセグメント番号を記憶するベース・レジスタに，各々のアドレス空間内のセグメント番号がセットされ（PBR には純粋プロシージャ部のセグメント番号が，ABR には作業用エリア部のセグメント番号がはいる．），各々のアドレス空間で走れば，自動的に作業用エリア部が区別されて使用される．

5020 TSS では，スーパバイザ，コマンド，ライブラリなどの各プログラムは，すべてこのような形式で使われる．

3.3 ダイナミック・リンキング

従来のシステムにおいては，あるプログラムとその中からコールされる種々のサブプログラムは，実行される前に，あらかじめリンケージ・エディタによって，すべて結びつけられ，それら全体が主メモリにロードされ実行されていた．いわゆる，スタティック・リンキングと呼ばれる方式である．これに対して 5020 TSS では，あるプロセスが必要となるプログラムのロードを，ファイルという単位ごとに，実行中に必要に応じて，ファイル・システムからダイナミックにロードし，それにあるセグメント番号を割り当てて，リンクをとるという，いわゆる，ダイナミック・リンキングの方式を採用している．

あるユーザの有するプログラム，あるいはデータの単位は，名前をつけられたファイルとして，ファイル・システムに登録されている．あるプログラム・ファイルから，他のプログラム・ファイルをシンボリックにコールしているとき，またはデータ・ファイルをシンボリックに参照したいとき，あるいは単に実行中に作業用のセグメントを割り当ててほしいというとき，コンパイラ，またはアセンブラのオブジェクト・プログラムは，そのプログラムの作業用エリア部(**3.2** で述べた）の中に，リンケージ・セクションと呼ばれるエリアを作成し，プロシージャ部からそこへインダイレクトに飛び越し，あるいは参照する方式をとる．

リンケージ・セクションの形を **Fig. 7** に示す．同図の〈……〉内はコンパイラ，あるいはアセンブラにより，その情報がセットされるエリアを示す．

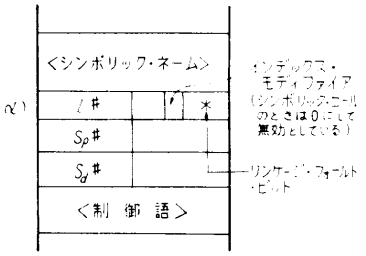

Fig. 7 Form of Linkage Section

いま，プログラムAから，ファイル・システムに登録されている他のプログラムBをシンボリックにコールした場合について考える．Aの作業用エリア部に，Fig. 7に示すようなリンケージ・セクションが作成され，シンボリック・ネームのエリアにはBが埋め込まれる．そしてさらに適当なコーリング・シークェンスの後に，そのリンケージ・セクション内の α (Fig. 7) に，インダイレクトに飛び越す命令が作られる．またパラメータのあるシンボリック・コールの場合には，そのパラメータのアドレスをセットするエリアがFig. 7の制御語の下に2語ずつ順にとられ，実行時にアドレス〈*si*, *li*〉がセットされる．他のセグメントをシンボリックに参照する場合にも，同じようなリンケージ・セクションがつくられる．

プログラムの実行が始まり，α 番地がインダイレクトに参照されると，第1回目の参照に対しては，Fig. 7 のリンケージ・フォールト・ビットがセットされており，そこが参照されることにより，金物的な割込みが発生する．そして制御は BFS (Basic File System) と呼ばれるスーパバイザに移行する．リンケージ・セクション内の制御語の中には，BFS に対する各種の情報がセットされている．BFS へ制御が移行してきたときに，BFS では

- プログラム・ファイルへの飛び越しか．
- データ・ファイルの参照か．
- 作業用セグメントの割当てか．

の区別を行ない，上二つの場合には，ファイル・システム内を，ある規則に従ってサーチし，目的とするファイルを見つけ出す（上記の例は，プログラム・ファイルへの飛び越しの場合である）．そして，そのファイルに対して，セグメント番号を割り当て，それをリンケージ・セクション内の S_p # としてセットする．S_d # には，プログラム・ファイルをコールした場合の，それに対応した作業用エリア部用のセグメント番

号をセットする．さらに，$l\#$ は割り当てられたファイルのセグメント内のロケーション (entry point) を示すものであり，ファイル・サーチの結果得られるものである．また同時に，いま割り当てた二つのセグメントに対応するセグメント・デスクリプタを SDT に登録する．

対象になっているファイルが，すでにセグメントとして割り当てられている場合には，ファイル・サーチを行なう必要はなく，単に，そのセグメント番号をリンケージ・セクション内の該当欄にセットすればよい．

また，作業用セグメント割当指定の際には，セグメントを一つ割り当てて，かつ，そのセグメント番号を，リンケージ・セクション内にセットすればよい．このような手順で，セグメントを割り当てた後に，リンケージ・フォールト・ビットをオフにし，制御を再びフォールトをおこした番地へもどす．すると今度は，自動的に $\langle S_p\#, l\# \rangle$ へ飛び越し（あるいは参照）することになる．すなわち，これをも含めて，2回目以降の参照に対しては，すでに割り当てられたセグメント番号により，参照が行なわれることになる．

3.4 共有セグメント

前にも述べたように，5020 TSS では，多くのプログラムが共有セグメントとして扱われている．ここでは，これら共有セグメントに対するセグメントの割り当て，スワップ，さらにセグメントの消去などについて述べる．共有セグメントに対する MT 上の構成は，LPDT を共通に使用するという方式が，一般の共有セグメントに適用されるものであるが，**Fig. 8** に示す構成も，特殊な共有セグメントに対して使用される．

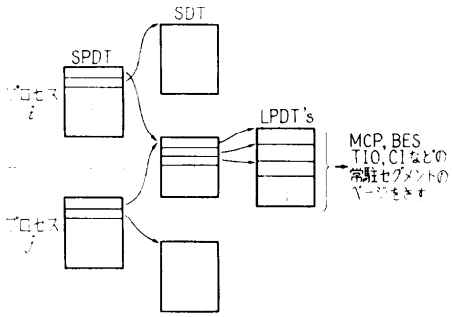

Fig. 8 Wired-down Common Segment

Fig. 8 で示した共有セグメントの特徴は，各プロセスごとに同じセグメント番号が割り当てられ，SDT をも共通に使用していることである．このことは，MCP (Master Control Program), BFS (Basic File System), TIO (Terminal I/O processor), CI (Command Interpreter) などのスーパバイザ・プログラムのように，すべてのプロセスに必ず使用される常駐共有セグメントに適用すれば，SDT が共通に使え，また，各セグメントの LPDT も，個々のセグメントの長さに見合った分だけを，とって常駐しておくことにより，MT の占めるエリアを節約することができるし，いろいろ便利な点も多い．この種の共有セグメント用として，数の小さい方からいくつかのセグメント番号を割り当てている．

さて，先に **Fig. 6** で示したような LPDT を共用するのが，一般共有セグメントの形式であるが，TSS の使用ユーザ数がふえ，セグメント数も増大してくると，これら共有セグメントに関するプログラムおよび MT も，スワップ・アウトしなければならない場合が多く生じてくる．このとき，共有セグメントを，他のプロセスからのつながり具合を無視して，勝手にスワップ・アウトすることはできないし，セグメントの消去に際しても，同様のことがいえる．

共有セグメントに関するこれら種々の問題点を解決するために，アクティブな共有セグメントに関する情報をたくわえておくために，ACFT (Active Common File Table) を設けた．さらに，個々のセグメントに対して，セグメント・デスクリプタを2語ずつ割り当て（したがって，実際にセグメント番号は，偶数番だけが使用されることになる．），その2語目をソフト的な管理情報として使い，共有セグメントの場合には，そこに ACFT 内の対応するエントリへのポインタを

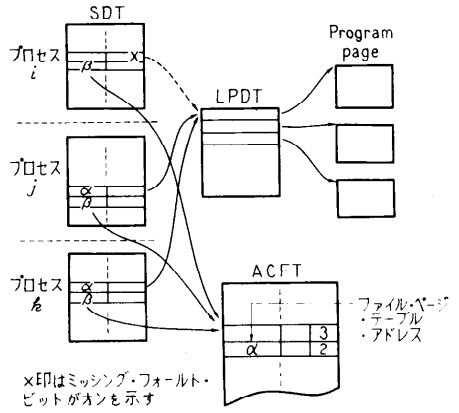

Fig. 9 Relation between Segment Descriptors and ACFT Entry for Common Segment

設ける（**Fig. 9**）．こうして，同一の共有セグメントをさすすべてのセグメント・デスクリプタと，ACFT 内のエントリを結びつけることにより，個々の共有セグメントの状況を，ACFT 内の情報に基づいて，制御を行なうことができる．

セグメント・デスクリプタの機構は，1語目には対応する LPDT がコアにあれば，その先頭アドレスをさしているが，LPDT がコアにない場合，それが共有セグメントでないときは，LPDT の2次メモリ上のアドレスがセットされるのに対し，共有セグメントのときには，2語目の “pointer to ACFT entry” の情報が有効となる（**Fig. 10**(a)）．

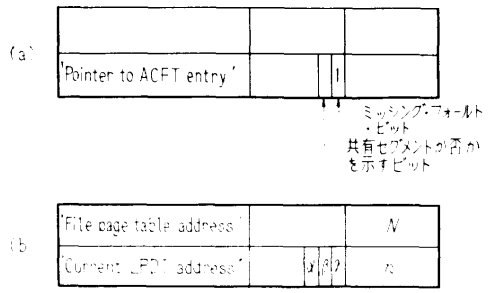

Fig. 10 Contents of Segment Descreptor and ACFT entry

ACFT 内の各エントリには，**Fig. 10**(b) に示すように，アクティブになった共有セグメントに関するセグメント割り当て，スワップ，消去などに対する有効情報が含まれている．各々を簡単に説明すると

- “file page table address”　これはこの共有ファイルに対するディスク上のファイル・ページ・テーブルのアドレスであり，ダイナミック・リンキングによって，新しくファイルが呼ばれたときに，それが，すでに他のプロセスによって，使われているか否かを調べる際に使用される．
- “current LPDT address”　この共有セグメントに対する LPDT が現在どこにあるかを示し，コアにあればコア・アドレスをスワップ・アウトされていればスワップ・アドレスをさしている．
- α　この共有セグメントの LPDT が，現在コアにあるか否かを示す．
- β　これは，この共有セグメントの LPDT がコアにないとき，ディスクにあるか，ドラムにあるかを示す．
- γ　この共有セグメントの LPDT が，スワップ・インあるいはアウトが行なわれたときに，そのI/Oが終了するまで，スワップ中であることをインディケイトしておくものである．
- N　現在この共有セグメントを使用しているプロセスの数を明示する．
- n　この共有セグメントの LPDT がコアにある場合，その LPDT と実際に結ばれているプロセスの数を示す．

以上述べたようなセグメント・デスクリプタと，ACFT 内の情報に基づいて，共有セグメントの割り当て，スワップ，消去が，どのように行なわれるかを述べる．

（1） 共有セグメントの割当て

あるプログラムからバーチャル・ネームをコールすると，BFS に制御が移り，そのプロセスが，すでにそのファイルを使用しているかどうかを調べる．もし，使っていれば，新しくセグメントの割り当ては行なわず，そのまま使用する．まだ，使っていなければ，ある規則に従ってファイル・サーチが行なわれ，そのファイルに対するファイル・ページ・テーブル・アドレスを得る（この過程は，共有ファイルであってもなくても，全く同じである．）．このとき，このファイルが共有ファイルであれば，すでに，他のプロセスによって，使用されているかをチェックするために，ACFT 内に，いま得たファイル・ページ・テーブル・アドレスと一致するものがあるかどうかを調べる．一致するものがあれば，セグメント番号を割り当てて，セグメント・デスクリプタを登録するときに，ミッシング・フォールト・ビットを立てておくとともに，一致した ACFT 内エントリへのポインタをセットしておく．こうしておけば，次に，このセグメント・デスクリプタでミッシング・フォールトがおきたときに，ACFT 内エントリへのポインタに基づいて，ACFT 内の ‘current LPDT address’ が求められ，共通の LPDT とつながる．さらにこのとき，この共有セグメントの使用プロセス数がふえたので，ACFT 内エントリの N を1だけふやしておく．ACFT 内に一致するものがなければ，この共有ファイルは，このプロセスによって，初めてコールされたわけで，ACFT 内に新しいエントリが登録される．

（2） セグメント・デスクリプタにおける ミッシング・フォールト

プログラムの実行中に，セグメント・デスクリプタでミッシング・フォールトがおこると，それが共有セグメントの場合には，フォールトをおこしたセグメン

ト・デスクリプタの2語目にかかれている ACFT 内エントリへのポインタに基づいて ACFT る調べ，対応する LPDT が，現在どこにあるかを調べなければならない．LPDT がスワップ中であれば，いま LPDT のスワップが行なわれている最中であるので，このプロセスは，そのスワップが終了するまでブロックされる．LPDT がコアにあれば，"current LPDT address" がそのアドレスをさしているので，セグメント・デスクリプタにこのアドレスをセットして，ミッシング・フォールト・ビットをオフにするだけで，LPDT を接続される．これにより，コアにある LPDT に接続されるプロセス数がふえたわけで，ACFT 内の n に1加えられる．LPDT がコアにないときには，この共有セグメントが最初に参照されたか，LPDT がスワップ・アウトされていたかのいずれかであり，いずれの場合も "current LPDT address" で示される2次メモリからスワップ・インを行なう (**Fig. 11**).

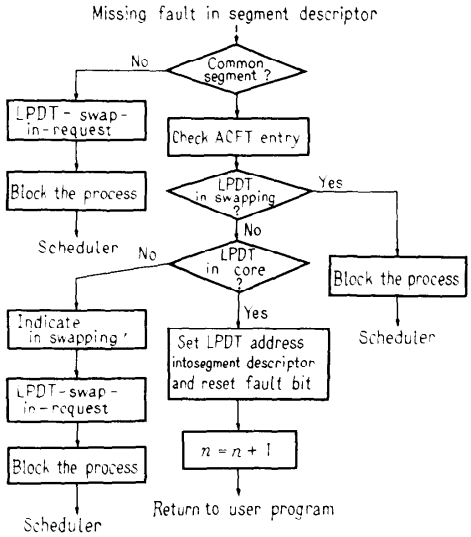

Fig. 11 Flow of Missing Fault in Segment Descriptor

(3) 共有セグメントのスワップ・アウト

共有セグメントでないセグメントに関するプログラム，あるいは MT のスワップについては，あまり問題はない．共有セグメントのプログラム・ページをスワップするときには，そのスワップ中に，他のプロセスによって参照される危険性があるので，各ページ・デスクリプタ内にスワップ中であるか否かのインディケータを設け，これがついている間に参照された場合には，そのプロセスをスワップ終了までブロックさせる．

共有セグメントの LPDT スワップについては，ACFT 内の情報に基づいて，他のプロセスからのつながり具合を考慮しなければならない．すなわち，LPDT をスワップ・アウトする場合，現在コアにあるその LPDT に，いくつのプロセスが実際につながれているかを調べ，もし，$n=1$，つまり，このプロセスのみにつながれているのであれば，$n=0$ として，この LPDT をスワップ・アウトする．このとき LPDT がスワップ中であることを，ACFT 内のスワップ中インディケータに明示しておく．もし，$n>1$ であれば，この LPDT には，他のいくつかのプロセスからもつながれているので，まだスワップ・アウトするわけにはいかず，単に，n を $n-1$ として，セグメント・デスクリプタのミッシング・フォールト・ビットを立て，このプロセスとのつながりだけを切断しておく (**Fig. 12**).

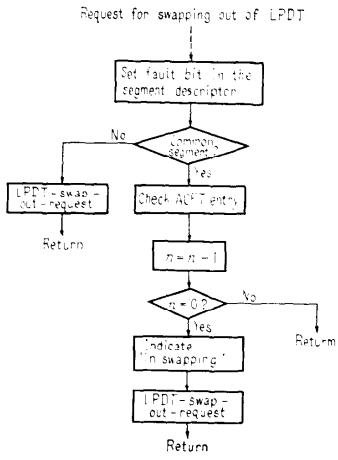

Fig. 12 Flow of Swapping Out of LPDT

(4) 共有セグメントの消去

ユーザのジョブが終了したときに，そのプロセスが，その間に使用していたすべたのセグメントを消去するとか，あるいは任意の時点で，個々のセグメントを消去する場合に，やはり，それが共有セグメントであるときには，注意しなければならない．すなわち，共有セグメントの消去は，それを使用していたすべたのプロセスから，そのような要求があるまで行なうことができない．これを制御するための情報が，ACFT 内

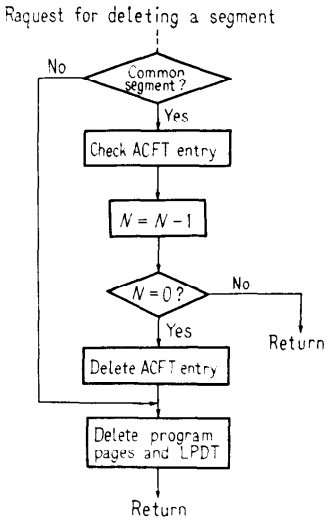

Fig. 13 Flow of Delating Segment

の使用プロセス数 N である．セグメント消去の要求があるごとに，N から1ずつ引いていき，0になったところで，初めて，そのセグメントのプログラム，およびLPDTなどを主メモリ，あるいは2次メモリから消去する．共有セグメントが消去されると，ACFT内から対応するエントリが除去される（**Fig. 13**）．

4. むすび

2次元番地付方式を採用した 5020 TSS のスーパーバイザの開発は，1968年3月で，その“核”となる部分の開発を終了した．

第1次計画が終了した現在，第2次計画として，次の目標に取り組んでいる．

（1） 2次元番地付方式の評価の問題

2次元番地付方式に関しては，種々の批判もあるが，それ自身非常に洗練されたアーキテクチャであると考える．われわれは，われわれの開発したシステムについて充分のデータをとり，それについてシステム・オーバヘッドなどを含めて，できるだけの解析を行ないたいと考えている．

（2） ソフトウェアの自己増殖の問題とソフトウェアの生産性向上の問題

TSS をベースとして，ソフトウェアの自己増殖，および生産性の向上が，どのように行なわれるか，また，行なえばよいかということは，非常に重要な問題であり，また，興味深い問題でもある．

（3） タイムシェアリング・システムの実用化

現在，TSS は国内では，まだ実験の規模を出ていないであろう．しかしながら，この方式は，近い将来，必ず新しい分野のコンピュータ・ユーテリィテイを作り上げる基礎的な道具となるであろう．

最後に，本研究の遂行に関しては，東京大学の高橋秀俊教授，後藤英一助教授，和田英一助教授，亀田寿夫氏，野口健一郎氏には，多くの討論会を通して，かずかずの有益なアイデアをいただいた．また，同時に，東京大学宇宙航空研究所の穂坂衛教授，大須賀節雄助教授からは，有益なご助言をいただいた．特に，大須賀助教授には，スケジューリング・アルゴリズムについて，新しい方式を提案していただき，現在の TSS に取り入れさせていただいた．また明治大学の稲富彬助教授にはシステム設計の段階で，いろいろご協力をいただいた．上記の方々に衷心よりお礼申し上げる．

参考文献

1) F. J. Corbató and V. A. Vyssotsky: “Introduction and Overview of the Multics System”, AFIPS Conference Proceedings, Vol. 27 (1965 FJCC) (Spartan Books. Washington, D. C., 1965) pp. 185～196, 同誌他四編, pp. 197～241.
2) J. H. Saltzer: “Traffic Control in a Multiplexed Computer System”, MAC-TR-30 (1966, Project, MAC).
3) J. B. Dennis: “Segmentation and the Design of Multiprogrammed Computer Systems”, JACM **12**, 4, pp. 589～602 (1965).
4) 益田，本林，高橋：“HITAC 5020 TSS のファイル・システム”，情報処理，**9**, 4, pp. 326～334 (1968).
5) 和田：“オンライン・コンピュテーション”，情報処理，**8**, 6, pp. 338～344 (1967).
6) 大須賀：“計算機システムの最適化制御”，情報処理，**9**, 2, pp. 88～97 (1968).
7) 嶋田，他：“HITAC 5020 TSS の概要”，他 12件，昭43 電気四学会連大論文集，分冊31，情報処理(1)，pp. 2956～2968.
8) 和田：“TSS 用通信制御装置のプログラム”，情報処理，**9**, 6, pp. 335—343 (1968).

（昭和 43 年8月6日受付）