U. D. C. 681.326.06

HITAC 8700 オペレーティング・システム

HITAC 8700 Operating System

堂免信義* Shingi Dômen　高須昭輔* Akisuke Takasu
日置敦* Atsushi Hioki　長谷川誠一* Seiichi Hasegawa

要　　旨

計算機システムの大形化はきわめて急であり，システムに結合される入出力装置や記憶装置の量的拡大も著しい。一方，利用形態の進歩も大きく，TSSやリモート・バッチも汎用システムに採用されている。またハードウェア自体も仮想メモリ（Virtual Memory），マルチプロセッシング機能など性能向上を目ざす革新が行なわれている。HITAC 8700はこれらの機能を備えた本格的大形システムであり，これを効率よく管理し，良質なサービスを提供するためにHITAC 8700オペレーティング・システムを開発した。同システムは各種の利用形態，処理形態を同一システムで統一的に処理すること，多数の装置を効率よく管理することなどのため多くの新しい機能を備えており，70年代を代表する新しいオペレーティング・システムである。

1. 緒　　言

1.1 大形システムの存在理由

一般に大形であることのハードウェア条件は，

（1） 高速演算機能
（2） 大容量メモリ
（3） 高速多数のチャネル
（4） 多種のI/Oデバイス

などを高信頼性で提供することであり，このような機能を多量に利用する利用者が増加する傾向にあると考えられる。大形導入の計画に際し，小形の複数化を選ばない理由は単に経済性ばかりでなく，**表1**に示すような各種の理由があるからである。

表1　大形と小形の比較

#	項　　目	小形の並置	大　　形
1	大量の小形ジョブ	可能であるが，負荷の配分が面倒である。オペレータが多数必要。分割損大。	可能，負荷配分不要，オペレータの省力化も可能。分割損少ない。
2	大形ジョブ	不可能（分割できない）	可　　能
3	OSの機能	単　　純	豊　　富
4	業務の集中と割込み	困　　難	容　　易
5	業務の種類	小形ジョブだけ	多様なジョブを受け付けられる。

1.2 大形システムに要求される機能

大形システムに要求される機能は**表2**に示すとおりである。小形コンピュータをパーソナル・コンピュータと呼ぶこともあるが，大形コンピュータはコミュニティ・コンピュータと称することができる。コミュニティに要求されることは，豊富な施設とそれらが共同利用できることである。

豊富な施設としては，デバッグ性能のすぐれた言語プロセッサと便利なファイル管理機能であろう。情報処理活動の中心はファイル処理にあると考えられる。ファイルの作成，更新，参照に関する機能の効果的な選定とその永続的な拡充が必要とされる。

利用形態の多様性が期待される。従来のバッチ中心システムから即時性と会話機能を備えた利用方式が具備されねばならない。またコミュニティに参加するユーザーの状態を記録したり，各種の資格審査をしたりする機能も必要となる。

表2　大形システムの存在理由と要求される機能

#	理　　由	要求される機能
1	大形ジョブの需要増大	大容量メモリ，高速演算，高精度演算，これらを活用するコンパイラ，管理プログラム。
2	省力化（小形システムの併設では負荷配分と操作がたいへん）	入出力量の削減，入出力操作の軽減，オペレーション業務の生産性向上，管理情報の採取。
3	情報コミュニティの拡大	大容量ファイルとその管理/アクセス法，実時間形/リモート・バッチ形/会話形などの処理形態。
4	情報の増大	ファイル/ライブラリの共用，ライブラリの管理機能，機密保持。
5	経　　済　　化	保持資源の分割損を縮小する。共用の効果を増大させる。

1.3 システムとしての8700・OS

HITAC 8700オペレーティング・システム（以後，8700・OSと略称する）は大形システムである。大形ソフトウェア開発の困難さが指摘されている今日，好んで大形化を志向するものではないが，存在理由と要求される機能をじゅうぶんに考慮し，それらを積極的に達成したものである。

システムを「特定の目的達成のための機能の配列」と定義するならば，8700・OSの目的は，

（1） 長時間での処理量の増大
（2） サービス形態に応じた応答時の確保

にある。ユーザーがこのシステムに期待する機能の本質は「情報の変換と検索」であろう。変換処理，検索処理に対し（1），（2）項の目的を果たすわけである。

一方，利用者から見ればこのシステムもより大きなシステムの一環であり，単なる道具に過ぎないだろう。道具全般に通ずる性質として便利で，使いやすく，高性能で，安全であることが望ましいが，8700・OSはこれらの性質を保持するとともに，多様性，拡張性，適応性に対してもじゅうぶん考慮されている。

* 日立製作所ソフトウェア工場

表3 特長的機能と設計のねらい

	特長的機能		前提			目標			
	大分類	中分類	多様性	拡張性	適応性	便利さ	使いやすさ	高性能	安全性
1	各種処理形態の統一的サポート	(1) バッチ形/実時間形/リモートバッチ形/会話形 (2) 同じコマンド体系 (3) ファイルも総合的に管理されている。	○			○ ○ ○	 ○ ○	○	 ○
2	可用性の向上	(1) マルチ/デュプレクスのどちらの構成も可能である。 (2) 障害に対処する機能がある。	○					○	○ ○
3	仮想メモリの活用	(1) メモリの使用効率が向上する。 (2) リエントラント・プログラムが作りやすい。 (3) プログラム間の保護が確実。 (4) 広いアドレス・スペースを利用できる。		 ○		 ○	 ○	○ ○	 ○
4	大きな拡張可能性	(1) ソフトウェア上は 16 MB までのコア・メモリを使える。 (2) モジュラリティを取り入れたシステムである。		○ ○					
5	すぐれた言語プロセッサ	(1) コンパイラがリエントラントである。 (2) リエントラントなオブジェクト・プログラムも作れる。 (3) FORTRAN は4倍長演算をサポートする。 (4) デバッグ用の機能が豊富である。				 ○ ○	 ○	○ ○ ○	
6	プログラム作成効率の向上	(1) 端末からもプログラムの作成と修正ができる。				○	○		
6	プログラム作成効率の向上	(2) プログラムの結合テストのために種々の機能がある。 (3) 利用者のプログラムをファイルに保存する。				○ ○		○	
7	強力なジョブ管理機能	(1) ジョブを緊急度と種別に応じてスケジュールする。 (2) システム資源を効率的に配分し，また過負荷がかからぬように制御する。 (3) オペレータ業務を軽減する。 (4) システム管理者/グループ管理者の概念を採用するなど，センタ運営の合理化に必要な機能を取り入れる。 (5) コマンドの修正追加ができる。			○ ○	○ ○ ○	 ○ ○	 ○ ○	
8	強力なファイル管理機能	(1) ファイルのカタログができ，ファイル指定が簡単にできる。 (2) 各種のアクセス法がある。 (3) ファイルに対する機密保護ができる。 (4) カタログを階層構造にしてグループ間のファイル共用ができる。 (5) ライブラリもファイルとして管理する。	○	 ○		○ ○ ○ ○	○	 ○	 ○
9	柔軟なシステムジェネレータ	(1) 構成に応じたシステムを作る。 (2) センター用のオウンコーディングを追加できる。 (3) システム・パラメータが設定できる。これらパラメータの一部はコマンドで稼働中に変更できる。			○ ○ ○			○	

2. 8700・OS の特長

2.1 利用者から見た新しさ

新しいものには取りえがなければならない。このシステムの新しさは，仮想メモリの積極的採用とそれを利用した各種処理形態の統一，4 CPU までのマルチ・プロセッサのサポートである。

世界的に見ても，仮想メモリ，マルチ・プロセッサをサポートし，会話形とバッチ形の利用を同一のコマンド言語体係で実現する最初のシステムである。

8000 シリーズのオペレーティング・システムである DOS, EDOS と比較すると次のような機能が新しく追加，強化されている。

(1) マルチ・プロセッサ
(2) 仮想メモリ
(3) 会話形処理
(4) ダイナミック・リンク
(5) リエントラント・コンパイラ
(6) リモート・バッチ
(7) マルチ CUP
(8) マルチ・ジョブ・ストリーム
(9) 可用性の向上
(10) 自動ボリューム認識
(11) ファイル・スペースの割当て
(12) 入力リーダと出力ライタ
(13) ファイルのカタログ機能
(14) 各種のスケジューリング
(15) 1コピーで共用される FCP(データ管理)

この中で，(6)～(11)については EDOS-MSO で，(12)についてはすでに EDOS で解決されている。

2.2 特長的機能

特長的機能は**表3**に示すとおりである。多様性，適応性，拡張性などの前提と便利さ，高性能などの目標が機能とどのように対応しているかも示してある。

2.3 モジュラリティと拡張可能性

拡張性，保守性を維持するためモジュラリティを盛りこむ必要性はだれしも認めるところであるが，具体性に乏しく精神的かけ声に終わる例も少なくない。8700・OS では，全プログラムを次の4階層に分け，徹底したモジュール化を実現している。

(1) プログラム……機能の単位
(2) フェーズ……ローディングの単位
(3) モジュール……アセンブリの単位
(4) ルーチン……最小単位

ことに最小単位のルーチンでは，アセンブラ・コーディングで150ステップ以下，ルーチンへの入口は先頭に限定する，としてモジュール性を良くしている。システム全体がこの階層で木構造を形成しており，ルーチンには一貫した番号が付されている。したがって保守性，拡張性は従来システムに比べると格段に向上している。

3. 四種の処理形態

バッチ形，リモート・バッチ形，実時間形，会話形を四種の処理形態と呼んでいる。前二者はスケジューリングを任意に行なうことができ，入力後は出力完了まで利用者の手を離れる。残り二者は，即時性と利用者の介入が実行中に常に必要である点で，バッチ形と異なっている。処理形態の差はスケジューリングと入出力の差にあると考えられる。

3.1 処理形態間の相違

（1） バッチ形とリモート・バッチ形

利用者から見ると入出力装置の設備場所がセンタ内にあるか遠隔地にあるかの相違だけである。リモート・バッチ形でも出力結果をセンタ内のラインプリンタに直接出力することができる。入出力に関しては両者間には大きな相違があり，システム側もそれぞれ別々にサポートしている。

（2） 実時間形と会話形

いわゆるTSSという言葉（ことば）はずいぶん広い範囲で用いられている。実際はリモート・バッチであるものを，システムがタイム・シェアで使用されているからという理由でTSSと称する場合もある。この論法でいくとバッチのマルチ・プログラミングも，システムがタイム・シェアで使用されてるいから，TSSの一種ということになるが一般に通信回線を経由しないとTSSとは呼ばないようである。8700・OSでのTSSは会話形処理のことを言うものとする。実時間形と会話形との相違が明確になればTSSの定義もおのずから明らかになる。

8700・OSでは次の条件を満たすジョブを会話形と呼んでいる。

（a） 端末からジョブを起動できる。

（b） 出力に応じて入力できる（いわゆる会話処理）。

（a）はシステム側にその機能がなければならない。（b）に関しては，システム側は論理的な入出力手段を提供するにとどまり，端末から起動されたプログラムに会話能力が付与されていなければならない。

（a），（b）両項を実時間形に適用してみると会話形との違いが明白になる。実時間処理はあらかじめセンタ側で起動され，一般には端末から起動されない。端末からはメッセージが入力される。メッセージは入力データにあたり，ジョブの起動とはなんら関係ない。また実時間処理は端末からの入力に応じて出力データが端末に送られ，端末側ではこのデータに基づき仕事が続行される。すなわち入力に応じた出力が原則でありその逆ではない。

バッチ形との対比も鮮明である。バッチ・ジョブは所望の出力を得るためにセンタから入力される。

会話形はまだ評価も定まらない形態であるが，基本的には（a），（b）両機能をシステム側で具備しておき，そのうえにすぐれた会話形言語をユーザー，システムの協力で順次開発していくことになる。

3.2 統一的処理

統一的処理とは，四種の処理形態を単一のシステムで同時にサポートすることである。8700・OSはシングル・プロセッサでもマルチ・プロセッサでも必要な資源があれば，バッチ形でも会話形でも実時間処理でも同時に多重処理することが可能である。同時性と並んで重要な特長は，四者間に統一した思想があり，コマンド言語もファイル構造やアクセス・マクロも会話形，バッチ形を通じて共通の仕様になっていることである。ファイル・カタログはシステムに唯一であり，会話形で作成したファイルをバッチ・ジョブや実時間ジョブからアクセスすることができる。

4. 仮想メモリ

4.1 基本的発想

仮想メモリの本質はプログラム・アドレスとメモリ・アドレスの分離にある。実メモリ方式では，ロードした場所のメモリ・アドレスに合わせて，プログラム・アドレスを変更しなければならなかった。仮想メモリ方式では，プログラム・アドレスは仮想アドレスに合わせるだけでよい。つまりプログラムは仮想メモリにロードされるのであって，コア・メモリ・アドレスとは全く無関係である。

一般に大きなプログラムは，コア・メモリにはいりきらないので全体を木構造にしている（図2の右側）枝Aの次に枝Bをコールすると枝Aは消されて枝Bがオーバレイされる。ところで枝を収容するだけのコア・メモリがじゅうぶんない場合は，さらに枝の分割に苦労しなければならない。

仮想メモリ方式では，オーバレイ構造をとらずとも，必要区画だけをハードウェアがオペレーティングシステムの力を借りて自動的にロードすることができる。プログラム作成時にコア・メモリの広さを考慮する必要は論理的にはなくなる。効率だけは考慮の対象として残る。また真に必要な区画だけがロードされて，メモリの使用効率が向上することも考えられる。区画としての単位は4,096バイトであり，1区画をページと呼んでいる。

ユーザー・プログラムはコア・メモリのどのページにロードされるか全く無関心でよい。仮想メモリとコア・メモリの対応は変換テーブルによりページ単位に行なわれる。変換テーブルはソフトウェアによりコア・メモリ上に作成され，ハードウェアが参照される。

4.2 仮想メモリの利点

（1） メモリ効率の向上

実メモリ方式ではプログラムを収容するにじゅうぶんの広さが連続して存在しなければならない。仮想メモリ方式ではページ単位に不連続エリアを拾い集めて活用できる。

（2） タスク間の保護が完全

変換テーブルはタスクごとに作成されるので，他タスクの暴走によって乱されることはない。

（3） リエントラント・プログラムの作成が可能

リエントラント・ルーチンは仮想メモリの助けを借りるまでもなく実現可能であるが，リエントラント・プログラムは困難である。その理由は一般にプログラムがオーバレイ構造になっているからである（図1参照）。タスク1（T_1）が枝Aを実行中にタスク2が枝Cをコールしても，実メモリ方式では枝A，Cは同じ場所にロードされるので，どうしてもシリアルにしか実行できない。つまりリエントラントにならない。しかし仮想メモリ方式ではコア・メモリさえあいていればどこでも使えるので枝A，Cが同時にロードされうる。仮想メモリではアドレス・スペースが広いので枝が別々の場所にロードされる。

（4） 広いアドレス・スペース

じゅうぶんに広いテーブルを始めから作ることができる。実際に使用する部分だけに実際のコア・メモリが割り当てられる。実メモリ方式ではテーブル・サイズの管理は常に頭痛の一つである。

（5） プログラムサイズの変動に強い

実メモリ方式では保有メモリを1バイトでも越えると全体が動

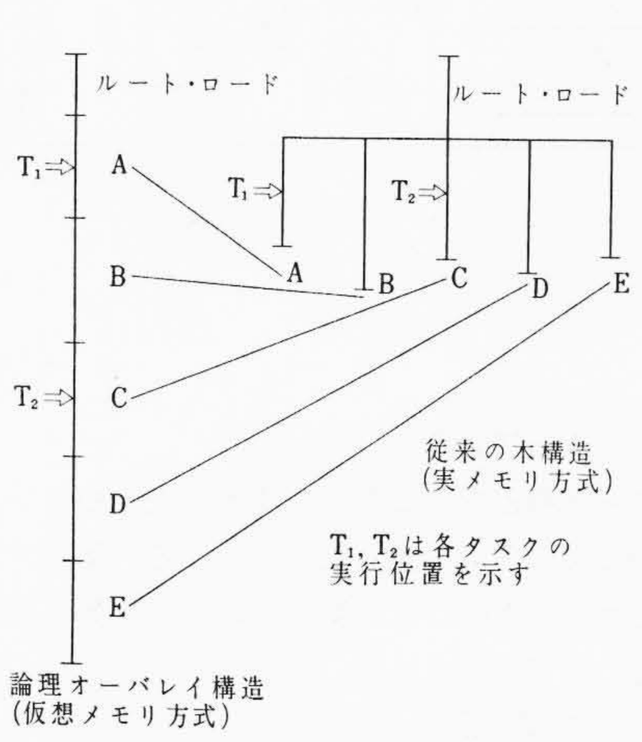

図1　リエントラント・プログラムと木構造

かなくなる。仮想メモリでは効率に多少の変動が生ずるだけであり，構造的にはそれほど敏感でなくともよい。

（6） 大形ジョブの実行

メモリ容量の制限から実行できなかったジョブでも，多少の時間をかけて仮想メモリ上で実行することができる。

4.3 8700・OSでの利用方式

システム・イニシエーションの初期と障害制御の一部を除き8700・OS全体が仮想メモリ上で動く。前節で述べた利点を最大限に活用する方式となっている。ただし言語プロセッサやユティリティ・プログラムは木構造をなし，おのおのの枝は100 KB以内のコア・メモリで効率よく動くようになっている。枝々は仮想メモリ上では異なるアドレス上にロードされるが，論理的には2個以上の枝が同時に走らないようになっている。すなわち排反関係の指定ができる。これを論理オーバレイ構造という。

この構造に基づき，コンパイラなどはページ単位ではなくフェーズ単位にロードされ，フェーズ終了後はそのメモリ・エリアを解放する。コンパイラはもちろんのこと，オブジェクト・プログラムもリエントラントにする機能がある。アドレススペースの広さを活用し，システムライブラリには一定のアドレス領域を割り当て，ライブリ上ではすべて絶対アドレス化されている。これによりシステム・プログラムのトラブル・シュートはかなり容易になる。またローディング時間も短縮される。

5. システムの構成

5.1 機 能 分 担

DOS, EDOSシステムに比べてオペレータの判断業務は大幅に減った。システム資源の管理と割当ては人間側からシステム側に移った。またディスク上のスペース割当て機能も，ユティリティ・プログラムを用いてプログラマが行なっていたものを，システム側がジョブ開始時に行なうことにした。これらはいずれも管理プログラムの機能強化により達成している。言語プロセッサは会話形言語もサポートする。ユティリティ・プログラムの分担に大きな変化はない。ソース・ライブラリの更新を端末から会話形で可能にする点が大きな進歩である。システムの構造は**図2**に示すとおりである。

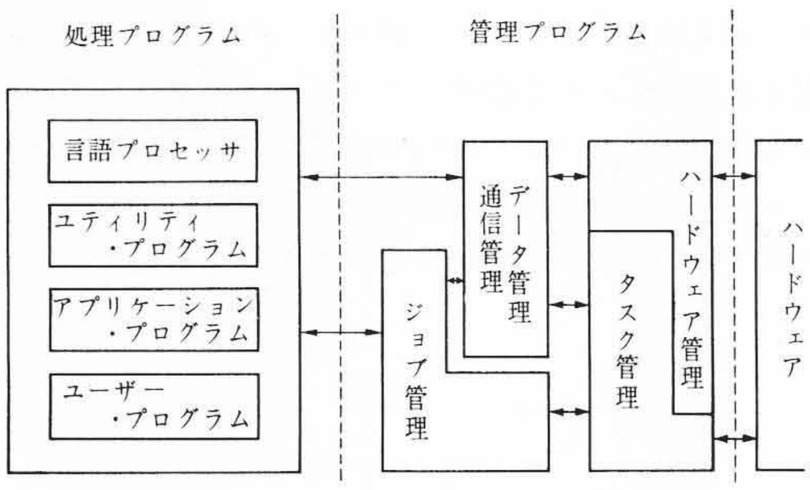

図2　システムの構造

（1） ハードウェア管理

ハードウェアの差をここで吸収する。入出力インターフェースの差，仮想と実の差もできるだけ吸収する。仮想アドレスで書かれた入出力コマンドはここで実アドレスに変換される。障害の検出と対策もここで処理される。

（2） タ ス ク 管 理

メモリ資源，CPU資源，ライブラリ資源の管理を行なう。またタスクの生成から死滅に到る管理も行なわれる。

ハードウェア管理，タスク管理ともかなりの部分が非タスク・モードで動く。残りのプログラムはすべてタスクとして動く。したがってマルチ・プロセッサの主要な問題点はハードウェア，タスク両管理で吸収される。

（3） データ，通信管理

ユーザー・タスクまたはシステム・タスクの延長上で動く。データのアクセス手段とファイル制御，スペース割当て，カタログ制御，回線制御などの機能を分担する。

（4） ジ ョ ブ 管 理

ジョブの入力と出力，スケジューリング，装置の割当て，コンソール制御，各種システム・ファイルの管理などを分担する。

（5） 処理プログラム

すべて非特権モードのタスクとして動く。シングル・タスクであればマルチ・プロセッサに関しては全く無意識でよい。タスク・ファミリのときは，同一ステップ内部のユーザー資源の管理を自ら行なわねばならない。処理形態間の差はほとんどない。

5.2 仮想メモリ配置

図3に示すとおりである。図中の木構造の部分が仮想空間の配置を示している。仮想空間はタスクごとに別々に設定されるが，0番地からM番地まではシステム空間として共用され，どのタスクからながめても同じものが出てくる。逆にシステム空間内からユーザー空間をながめると，その時点にそのCPUで走っているタスクしか見えない。つまりシステム空間は一重でユーザー空間は多重なのである。システム空間には全ユーザーから共用されるプログラムがはいる。

ユーザー空間どうしは互いに干渉しないが，ユーザー空間とシステム空間は互いに参照することができる。ユーザーの暴走からシステムを防御するものがリング保護である。リングレベルは0が最も高く6が最も低い。低い所から高い所を参照すると一般にリング保護エラーになる。ユーザー・プログラムはレベル5，6で走るので0～4にあるシステム・プログラムを破壊するおそれはない。

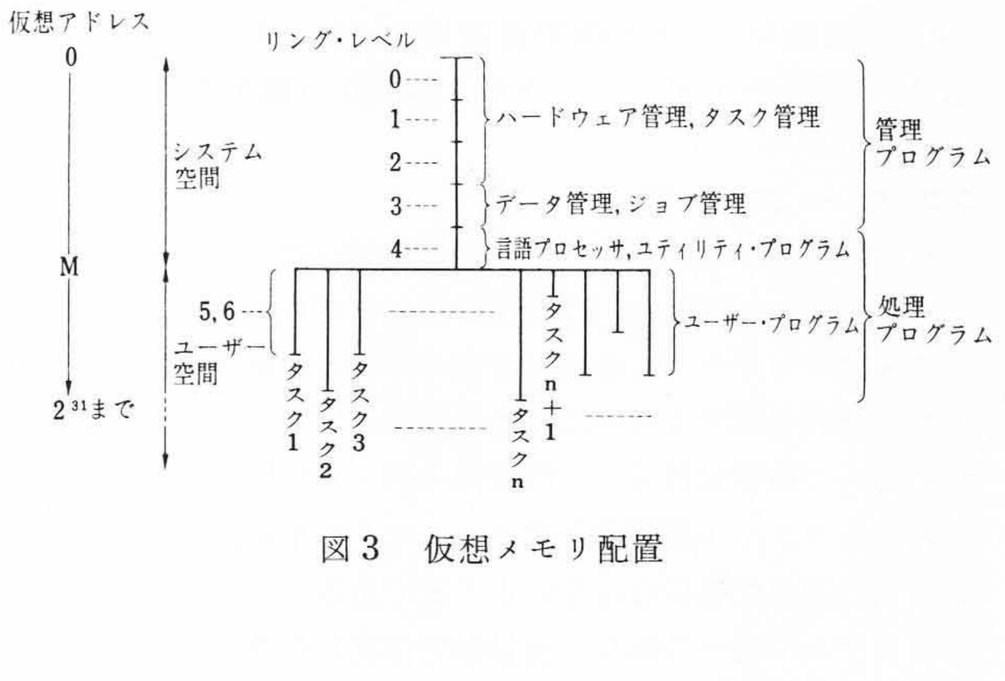

図3　仮想メモリ配置

6. 資 源 の 管 理

資源の管理は複数ユーザーがシステムを競合して使用する場合に必要となる。単一ユーザーが全システム資源を独占して使用するならば，資源の利用配分はユーザーにほぼ任せることができる。きわめて単純な競合であればユーザー間で協定したり，オペレータの判断に任せることもできる。しかし，いわゆる大形と銘うたれるほどのシステムではなんらかの方法でユーザーやオペレータではなくシステム側で管理する必要がある。資源とは，メモリ，処理装置，チャネル，ディバイスなどのハードウェア資源とプログラムやファイ

ルなどソフトウェア資源を言う。ここではおもにハードウェア資源について記述する。

資源管理の最も簡単な方法は，同時に登場するユーザーの数に合わせてあらかじめ資源を分割しておく方法である。メモリについてはよく採用される方式であり，いわゆるパーチション（Partition：分割）である。

パーチション方式は簡明さで非常にすぐれているが，

（1）　ジョブのタイプと大きさが截然(せつぜん)と分類できて，

（2）　負荷の変動が少ない。

というような例でないとやや効率が悪くなる。システム資源全体を時々刻々に変わる使用状況に応じて正確に把握(はあく)し，新しい要求には必要分だけを割り当ててやれば，どんな種類のジョブにも負荷変動にも対処することができる。しかし問題はそう簡単ではない。単に対処するだけではなく，デッドロックを回避し，効率よく資源配分を行なわねばならない。そもそも「効率よく」とはなんだろうか。CPU効率を第一義の目安とした時代もあった。そこではCPU効率の向上，すなわちアイドル率の低下を資源管理の最大目標とした。最近の大形システムのように大容量ファイルの接続が常識となっている場合はCPU効率だけに着目するわけにはゆかず，むしろ周辺装置を忙しく働かせることを考えねばならない。

純粋に経済的な見地からは単位時間あたりの使用料を最大にすることが効率よい管理であろう。すなわち高価なものほど，遊休させずに忙しく利用する方式である。しかしユーザーが期待するものはスループットの実績である。時間あたりの処理量である。

処理量とはなんだろうか。事務計算と科学計算を量的に比較することはできない。むしろ処理能力は，あらかじめ仮定したジョブ群を処理するための所要時間によって測られる。ところが一日のジョブ構成は日々変動し，きょうの処理量ときのうの処理量とは件数でしか比較できない。したがってシステム自身が処理量を測定し，その最大化をねらって資源管理することは一般的には困難である。むしろシステム運営の責任者の意志に沿って資源管理を行なうほうが現実的である。資源管理はOSの中心課題であるのでやや詳細に論じてみたい。

6.1　処理形態間の資源配分

会話処理とバッチ処理の並行処理では，バッチ対会話のCPU使用率その他の割当て比率を決め，会話形にかたよりすぎぬようにする。会話形の応答時間短縮のためバッチを無制限に犠牲にするシステムもあるが，8700・OSでは会話形はターン・アラウンド時間短縮のためのサービスであり，処理の中心はバッチ形にあるとしているので，バッチ処理能力の保証に重点が置かれている。もちろん比率を適当てに指定することにより，会話中心のシステムにすることもできる。実時間形をバッチ形に含めて考える。

会話形資源の制限は次の項目に対し指定することができる。

（1）　アクティブ端末の数

（2）　CPU占有率

（3）　アクティブ会話タスクの数

6.2　専有と共用

共用可能な資源は，自分自身への書込みのないプログラムや参照だけのファイルなどである。チャネル，デバイスなどもその部分は分割して専有されるが全体としては共用可能と言える。それに対して書込み変更の対象となる資源は独占専有して使わねばならない。参照中の他者がいると変更途中の誤った情報を与える危険性があるからである。

共用とは同時に二者以上が使用できることである。会議室のいすはだれでも利用できるが一時にひとりしかすわれないので，この意味では共用不可である。

共用不可の資源に対しては専有を宣言させる。あいていれば専有させ他者が専有していれば待ち行列に載せて待たせる。この機能を実現するためにENQマクロ，DEQマクロがある。二者が互いは相手が専有している資源を待つときに生じるデッドロックを回避するために，複数項目の資源に対し一度にENQすることが許されている。

装置割当ても一時に1台ずつでは容易にデッド・ロックを生ずるので，ジョブ・ステップの開始時に必要な装置を一度に割り当てる。割当て不能ならばそのジョブ・ステップを装置待ち行列上で待たせる。ジョブ・ステップとはジョブの構成要素であり，FORTRANのコンパイル，目的プログラムの結合，実行などがおのおの1ステップとなる。この方式では次のステップまで専有する装置があるときはデッド・ロックの可能性が生ずる。これを回避するため，装置待ち行列にジョブ・ステップを載せるとき，そのステップの保有する装置を待っているステップが先にあれば，まずその装置を解放してから装置待ち行列に載せる。

6.3　プログラムの共用

プログラムの共用はメモリ・スペースの節約，ロード時間の短縮の2点の効果がある。共用には同一仮想空間で共用する場合と異なる空間で共用する場合がある。すべてのシステム・プログラムやタスク・ファミリ内で共用するユーザー・プログラムは，それぞれ同一仮想アドレス上で共用されるので，実メモリの場合と同じ方式と考えてよい。

実メモリ方式では作業用エリアは動的に確保され，その起点はベース・レジスタによって指定される。タスクが異なれば作業エリアの場所も異なる。一方，仮想メモリ方式ではアドレス変換テーブルを別にすることにより，同一仮想アドレスの作業エリアを別々のコア・メモリ上に置くことができる。作業用エリアを静的に確保できるとも言えよう。したがって作業場所を指(さ)すアドレス定数をプログラム内に保有することもできる。実メモリ方式ではまず考えられないことである。**図4**は異なる仮想空間上の共用を示している。

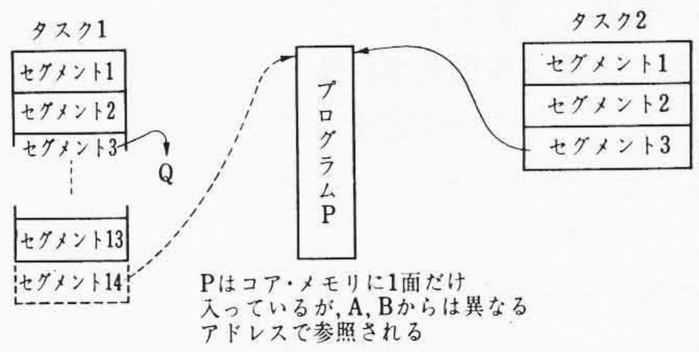

図4　プログラムの共用

タスク2ではプログラムFを3番目にコールし，セグメント3，として使っている。そこへタスク1からPをコールすると，タスク1はセグメント3を別のプログラムQに使っているのでPをセグメント3としてはコールできない。セグメント14としてコールすることになろう。セグメント番号は仮想アドレスの一部なので，プログラムPは単一のコア・メモリ上の単一の実体にもかかわらず，両タスクから異なる仮想アドレス上で参照されることになる。したがってPはP内の場所を参照するアドレス定数をP内部に保有することができない。そのためPSECT（Private Section）をタスクごとに設定する。PSECTはアドレス定数や作業用場所を収容するために使われ，COPYマクロにより仮想空間上に写しが作られる。写しの置かれる場所はレジスタ経由でユーザーに知らせる。

この方式はめんどうなようであるが，不特定多数のユーザー間で共同する最も一般的な方式である。

6.4 各種のスケジューリング

ジョブ，タスク，メモリおよび入出力装置のスケジューリングは相互に関連し，スループットに大きな影響を与える。実メモリ・システムでは，メモリは入出力装置と同じくジョブ起動時に割り当てられ，終了まで保持される。ロールイン，ロールアウトの導入により，実メモリ方式でもジョブの実行途中でメモリ割当てが更新されうることになるが，それほど忙しい更新ではない。いったん割り当てられたメモリは，参照の頻度(ひんど)にかかわらず最後まで専有されてしまう。仮想メモリ方式では，コア・メモリはチャネルと同じようにダイナミックに利用することができる。一方，コア・メモリ容量がシステム・スループットに多大の影響を与えることは早くから知られた事実である。ここに工夫しだいでは資源の使用効率を大きく向上させうる手段が与えられたわけである。

6.4.1 ジョブ・スケジューリング

バッチ・ジョブはセンタのカード・リーダその他の装置，遠隔端末の入力装置からシステムに投入される。会話形ジョブは端末から直接起動される。

バッチ・ジョブの受付けは無条件に行なわれるが，その起動は，

（1） ジョブ・クラス

（2） ジョブ・プライオリティ

（3） 必要資源のあきぐあい

により規制される。ジョブ・クラスはA～Zの26クラスあり，クラスごとに仮想メモリ，CPU時間，入出力量などの上限とジョブ取出し比率を指定することができる。ユーザーはクラスをジョブ・カード上で指定する。ジョブ・スケジューラはこの取出し比率に応じてクラスを選択する。取出し比率の高いクラスのジョブは頻繁(ひんぱん)にサービスを受ける。システム運営者は取出し比率を適正に設定することにより，Aを特急Bを普通などと指定し，専有資源量の少ないジョブには短いターンアラウンド時間でサービスすることができる。

クラスの概念は計算機資源の部門別割当てにも役立つ。計算機時間が不足状態にあるとき，部門Aが努めて節約を図っているのに部門Bが浪費して困るという苦情は，A，Bに同一の取出し比率を与えることにより解消される。同一部門内の優先処理にはジョブ・プライオリティを指定することにより実現できる。

入出力装置，仮想メモリなどの資源はジョブ・ステップ単位にまとめて割り当てられる。

6.4.2 タスク・スケジューリング

ジョブ・スケジュールの関門を通過したジョブはステップ単位にタスクとして実行される。タスクはCPU実行権，コア・メモリおよびチャネルの割当てを受ける単位である。タスクは連続的処理の過程で事象を待ったり割込みを受けたりするのに便利な概念があるのでシステム入出力などのシステム側の業務もタスクの形態をとっているものが多い。これらをシステム・タスクと呼んでいる。ユーザーのジョブ・ステップから発生するタスクはマルチ・ジョブの環境下では一般に複数個存在する。これをマルチ・タスクと称する場合もある。通常のジョブ・ステップからはただ1個のタスクが発生するが，通信量の多い実時間処理では同一ステップ内に複数個のタスクを必要とする場合がある。DOS, EDOSではこの場合の複数タスクをマルチ・タスクと言っているが，8700・OSではタスク・ファミリと呼んでいる。

タスク・スケジューリングの関心事は次にどのタスクを選択して実行するかにある。選択に際しての方策は次のとおりである。

（1） 優先度に応じた応答時間の保証

（2） システム効率の向上

（3） サービスの適正な配分

（4） 過負荷の排除

この方策を実現するためタスク・キューをいくつかのプライオリティ・レベルに分ける。高いプライオリティのタスクを先にサービスする。会話形への集中を避けるためCPU時間比を監視する。特定タスクでのCPUの長時間専有を排除するため，一定時間でサービスを中断する。この時間をタイム・スライスと言う。バッチ形にも適用される。

タスクの状態は図5に示すとおりである。ラニング中にタイム・オーバになったタスクはブロックに移る。端末出力中のタスクもブロックになる。ブロック要因が解消されても直ちにページングには移行せず，ペンディング状態でシステム負荷の調整を図る。過負荷状態に陥るのを防ぐためである。

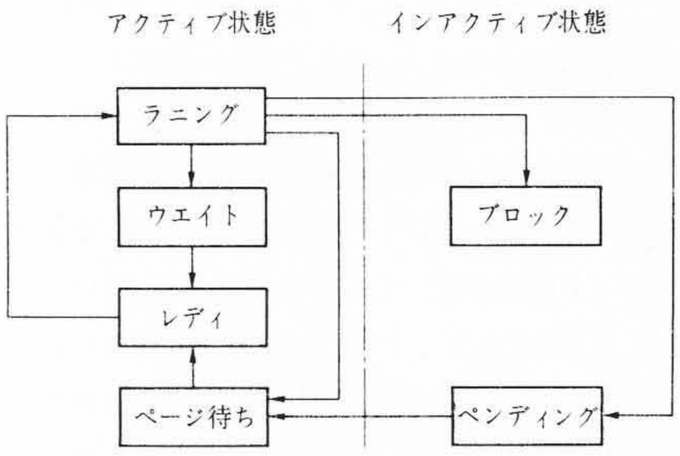

図5　タスクの状態とその移行

6.4.3 メモリ・スケジューリング

従来の実メモリ・システムではあまり技巧的なスケジューリングの対象ではなかったコア・メモリが，仮想メモリ方式ではスケジューリング技術上に大きなテーマを提供することになった。本来はメモリ・サイズの制限を緩和する，すなわちスケジューリングを楽にするために作られた仮想メモリ・システムでメモリ管理が複雑になるとは，一見皮肉な現象である。リソース・アロケーションに関する論文の過半はメモリ・スケジューリングを論じている。ユーザー側の負担がシステム側に転嫁されたともいえる。オン・デマンド・ページングの効率が問題とされ，できるだけ必要なページを先取りしておくプリページングが検討された時期もあるが，真の問題点は過小メモリでは有効なページが容易にスワップ・アウトされることである。いずれにしても，コア・メモリは有限であるから，新しい情報を受け入れるためには古い情報のページを追い出さねばならない。

8700ハードウェアにはページ・テーブル上にそのページを参照したとき，書き込んだときにそれぞれセットされるビットがある。参照ビット，書込みビットと呼ぶ，このビットを定期的にスキャンしリセットすることによりページに対するアクセス頻度との傾向を知ることができる。

ユーザー・タスクの保有するページは最も古く参照されたものから順にチェーンしておく。追出しが必要になったときはこのチェーンの端から追い出す。これをLRU(Last Recently Used)法と呼んでいる。LRU法でのスワッピング頻度(ひんど)が高くなり，過負荷状態にあると判断されたときは，最低プライオリティのタスクをブロックしそのタスクの保有する全ページを追出しの対象とする。

7. ファイルの構造と機能

ファイルに関する第一の機能はメモリと外部配置間の情報の転送である。第二はファイルそのものの参照権や更新権の管理である。

第三はランダム・アクセス・ボリューム上のスペース割当てに関する機能である。

7.1 カタログの有効性

カタログの目的はファイルをその名前で参照することである。カタログ自体もランダム・アクセス上のファイルとして存在し，そのカタログ・ファイルの中味はファイル全般の住所録と見ることができる。ファイル名称とそのファイルが存在するボリューム番号とが対になって登録されている。ユーザーはファイルとボリューム，デバイスとの対応に関し神経を使う必要は全くない。単にファイル名称を指定するだけで，もしそのファイルがオンラインであればそのまま，オフラインであればボリュームのマウント指示が出され，そのあとでデバイスが割り当てられる。

ファイル名称は1～8文字の文字，数字列からなる単純名称を，ピリオドで連鎖したものである。ピリオドの数は階層の深さを示し，ファイル名称を階層に対応してつけられるようになっている。

このような名称付けはユーザー側に便利なばかりでなく，システム側にとってもカタログ・ファイルを木構造で管理できる利点がある。本来ファイルというものは追加，削除が頻繁に行なわれるので木構造で管理すべきものである。

カタログのもう一つの機能は機密保持と共用の制御である。ファイルの所有者や参照権を有するユーザー名がカタログ内に登録されている。参照権のないユーザーからのOPENは拒否される。

7.2 ファイル・スペースの割当て

DTFカード上のSPACEオペランドで自由に指定でき，

（1） 最初にスペースを割り当てる初期割当て機能
（2） 割り当てられているスペースを拡張する拡張割当て機能
（3） 割り当てられているスペースを解除する割当て解除機能
（4） 割り当てられているスペースのうち，未使用の部分を解放する未使用領域解放機能

などがある。初期割当て機能にも，割当てずみスペースから切出しを行なう2次割当て，連続した1エクステントを水平方向に分割する分割シリング割当ておよびそのほかの一般割当てがある。

割当て要求量は，シリンダ，トラック，ブロック，ページの単位で指定される。割当て上の重要な関心事にエクステント数がある。物によってはどう分散していてもかまわないが，実時間ファイルや分割シリンダ割当ての対象にする予定のファイルはできるだけ集積していたほうがよい。1～29エクステントまでを4段階に分けて指定することができる。私用ボリュームに対してはトラック・アドレスを指定して割り当てることが可能である。

スペース割当ての技術上の問題点は指定されたエクステントの分散に沿ってどのようなアルゴリズムで割り当てるかである。

7.3 アクセス法

順アクセス法，直接アクセス法，索引順アクセス法に大別される。ページ・アクセス法関係の使用はシステム・プログラムに限定される。ユーザーに解放されるアクセス法はQSAM，BSAM，DAM，QISAM，BISAMの5とおりである。これらのアクセス法はリエントラントになっており，複数人のユーザーが同時に使用しても，ルーチンはシステム空間に1面だけしかロードされない。

BISAMとDAMは実時間で効率よく使えるように，タスク・ファミリ内でのブロックの排他的制御やファイル制御ブロックの共同を許している。

8. 端末との通信

8.1 アクセス法の一種として

一般に外部記憶装置上のデータはファイルという概念でとらえられ，このデータの転送に関してはアクセス法が確立されている。カード・リーダやラインプリンタに対するアクセス法も存在する。当然遠隔端末上のデータにもファイルの概念を適用し，アクセス法として統一した処理を適用すべきであるという考え方が生まれる。8700・OSもこの思想に沿い，当初はデータ管理の一環として端末との通信制御を設計していたが，一般周辺機器とは次の点で，顕著な違いがあるのでついに通信アクセス法としてデータ管理から分離した。

（1） 端末の動作とプログラムの動作が非同期である。端末のオペレータの意志により入力が開始される。データの発生の時点に関してプログラムは関与できない。
（2） 実時間処理では処理の対象となる端末台数が周辺装置に比べて非常に多い。
（3） 入出力の制御方式が異なる。これはハードウェア上の問題であるが，信号方式と呼ばれる特殊なインターフェースがあり，一般周辺装置と同じタイプで扱うことができない。

とりわけ(1)の差が重要な意味を有し，逆にこの性格をもつ装置を8700・OSでは通信アクセス法の制御の対象としている。それはチャネル直結の端末やデータ交換装置(DXC)である。

アクセス法はBCAMとCCAMである。BCAMではユーザーは直接回線にアクセスする。CCAMでは端末のデータにアクセスし，端末自身は管理プログラム側の制御下にある。BCAMは実時間処理を，CCAMは会話処理をその主要な用途にしている。

9. その他

9.1 運転形態

マルチ・プロセッサ構成とデュプレックス構成を許している。前者が1システムであるのに比べ後者は2システムでファイルだけを共用する。マルチ・プロセッサ構成では各CPUは完全に平等ではなく，システム・タスク業務を特定CPUにくくりつけることができる。入出力操作も特定CPUで処理させることができる。業務を分担するほうが仮想メモリおよびバッファ・メモリの効率を高める。

9.2 課金

課金の対象となる情報をジョブ単位に抽出し，ディスクまたは磁気テープにダンプする。これにどんな料金を割り当てるかはシステムの運営者が決める。課金はシステム資源を有効に使うための唯一の方策である。需要と供給という経済原則が課金により計算機システムに導入される。運営者は料金付けを巧妙に行なうことにより，スループットを向上させるような資源利用へ利用者を誘導することができる。

10. 結言

通信管理，障害対策，運転形態，課金などの重要事項が紙面の都合で詳述できなかった。また開発体制や管理面の問題も割愛した。システムは現在収束段階にあり，一般ユーザーに使用される日も間近い。これだけのシステムが無事ここまでこられたのも，諸先輩の助言と激励のたまものでありここに厚く感謝したい。完成後も油断することなく機能，性能の改善に日々努力したいと思う。