HITAC 8700/8800 オペレーティング・システム (OS 7)†

大西 勛†† 秋田英彦†† 堂免信義†† 金子和政†††

Abstract

Several problems on design and implementation of the large scale operating system (OS 7) are discussed. OS 7 provides multi-virtual space facility and the support for multi-processor system with any combination of HITAC 8700 and 8800 up to 4 CPU's.

This report emphasizes discussion of the virtual memory scheduling algorithms and the task scheduling algorithms for the 8700-8800 multiprocessor system.

It also presents the algorithms used to control the shared program and the command system which has the command definition facilities and covers batch user commands (job control language), TSS user commands, center operator commands and remote batch operator commands. The overall structure of OS 7 is also presented.

1. まえがき

OS 7 は超大型電子計算機 HITAC 8700/8800 用の汎用オペレーティング・システムで，仮想メモリ機能と多重プロセシング機能を備えている．

多重プロセシング機能は最大4台まで CPU が接続可能である．HITAC 8700 と 8800 はソフトウェア的に完全に両立性があるばかりでなく，主記憶装置，入出力処理装置を両者で共用することができるので，8700 と 8800 の両方が混在した多重プロセッサ構成も可能である．

また，広範な利用形態に応じるため，クローズ・バッチ処理，オープン・バッチ処理，リモート・バッチ処理，TSS 処理，実時間処理のすべての形態を同一システムで，統一的に処理できるよう考慮が払われている．ここで統一的とは次のことを指している．

（1） 必要な資源さえあれば，バッチ処理も TSS 処理も実時間処理も同時に多重処理ができる．

（2） 同一のコマンド言語（ジョブ制御言語）を各処理形態間で共通に使用できる．

（3） 同一のファイル・システムを使用できる．ファイル構造もアクセス法もファイル・カタログも共通であり，バッチで作成したファイルを TSS で修正したり，実時間処理でアクセスしたりできる．

（4） 同一のプログラム・ライブラリを各形態間で共通に使用できる．

また，オープン・バッチ処理は東京大学大型計算機センターの HITAC 5020 E の運転経験からジョブの入出力オペレーションのセルフ・サービス制のアイデアが同センターより出され，開発したものである[2),3)]．特に出力はディスク記憶装置上にジョブの出力を貯えておき各ユーザが自分のユーザ ID のセン孔されたプラスチック製のトークン・カードを読取り機に入れることにより，対応するライン・プリンタまたはカード・セン孔機から自分のジョブの結果を取り出すことができる．

2. OS 7 の特徴的機能

2.1 仮想メモリ

2.1.1 仮想空間の構造

OS 7 の仮想空間は Fig. 1 のように，先頭の0番地～M-1番地が全ユーザに共通の空間（システム空間と呼ぶ）で，M 番地～2^{31}-1 番地が各ユーザに固有の空間（ユーザ空間と呼ぶ）になっている．またユーザ

† HITAC 8700/8800 Operating System (OS 7), by Isao Ohnishi, Hidehiko Akita, Shingi Domen (Software Works, Hitachi, Ltd.) and Kazumasa Kaneko (Hitachi Software Engineering Co., Ltd.)

†† 日立製作所ソフトウェア工場システムプログラム部

††† 日立ソフトウェアエンジニアリング株式会社

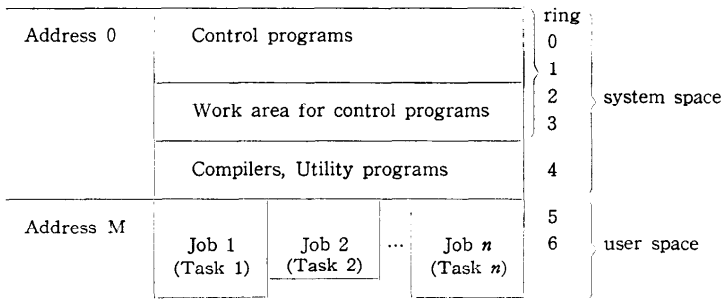

Fig. 1 Virtual space organization in OS 7

空間は同一アドレスが各ユーザ毎に異なる実メモリにアップされる多重仮想空間構造になっている．他のOSで，一つの仮想空間上で全ユーザのタスクが実行される方式を採用しているものもある[10]．これはアドレス変換テーブルが1個ですむため，メモリが節約できて取扱いも簡単であるが使用できるアドレス・スペースが充分広くなく，またメモリ保護にストーレジ・キー等を利用するため，ジョブ多重度を 15 程度にしかできないという欠点がある．これは TSS を行なう際に問題となる．多重仮想空間構造の場合にはアドレス変換テーブルを各ユーザ毎に持ち，他のユーザの空間とは一切重ならないようにアップして，メモリ保護を行なっている．理論的には多重度はいくらにでもすることができる．またユーザ当りのアドレス・スペースは 2^{31} バイトまである．メモリ保護についてはOS 7ではこの他にリング保護機能がある．これは各セグメントに7段階のリングを与えることができ，下位のリングにあるプログラムは上位のリングの領域に書き込むことができない．この機能を利用してシステム空間のメモリ保護を行なっている．リング 0〜3 は管理プログラム，リング4はコンパイラ，ユーティリティ・プログラム，リング 5〜6 はユーザ・プログラムに割当てられている．このリングの壁によりトラブル・シュートの際の調査対象範囲が狭くなり，対策が容易になっている．このような多重仮想空間を構成するためにアドレス変換テーブルは Fig. 2 のように作られている．

システム空間用のサグメント・テーブルとページ・テーブルはシステム中に唯1組のみ存在し，すべてのユーザで共用されている．Fig. 2 のようなベース・テーブルの利用により，ユーザ間で共用される空間は空間そのものはもちろんのこと，アドレス変換テーブル（セグメント・テーブルとそれからポイントされるすべてのページ・テーブル）もメモリ上に1組あればよいことになり，メモリの節約になっている．

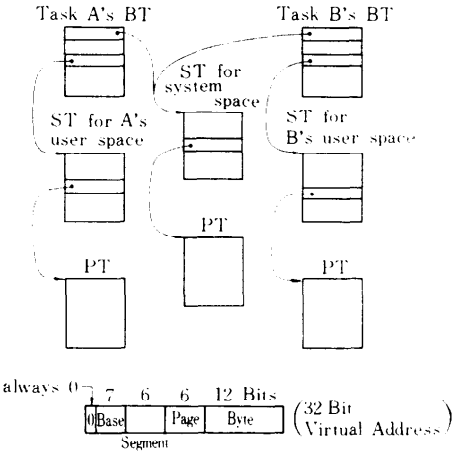

Fig. 2 Address translation tables and virtual address format

2.1.2 メモリ・スケジューリング

仮想メモリ・システムでは主メモリに対する要求はマッピング・フォールトという割込みの形式でシステムに伝えられる．これに対し，空きの主メモリがあればそれを与えるが，無いときには現在使用中の主メモリのページの中から適当なページを選択し，これを必要ならスワッピング・ドラムに書き出して，主メモリを明け渡し，これをマッピング・フォールトの起ったページに割当てなければならない．この「追出しページの選択アルゴリズム」がメモリ・スケジュール上最も重要である．選択アルゴリズムの目的を「マッピングフォールトの発生回数を最小にすること」と定義すると，「最も遠い将来に参照されるページを選択する」ことが最良のアルゴリズムになる．しかし，これは実用上は実現不可能である．そこでプログラムの一般的な性質から「より遠い過去に参照されたページの方が近い過去に参照されたページより，近い将来再び参照

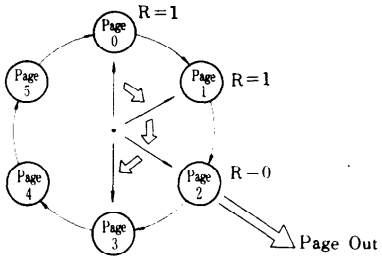

Fig. 3 First In Not Used First Out (FINUFO) Algorithm

される確率は低い」という仮定をおき，「一番遠い過去に参照されたページを選ぶ」LRU (Least Recently Used) 法を用いることが多い[1]．ここではこの仮定をLRU の仮定と呼ぶことにする．しかし厳密な LRU 法も実現は難しいので，OS 7 ではこれに比較的近い FINUFO (First In Not Used First Out) 法[7]を用いている．これは「はじめに入って使われていないページから先ず追出す」という意味である．次に FINUFO 法について詳しく説明する．

Fig. 3 のように追出し可能なページは環状リストになっている．

(a) 主記憶上の各ページには R (refer) ビットと C (change) ビットがついている．はじめは R=C=0 でそのページ内の1バイトでも参照されると，R=1 となり，また書き込みが行なわれると C=1 となる．

(b) 追出し可能ページ・リストにはポインタがありそれがどれかのページをさしている．

(c) 追出しページの選択をする必要が出来たら，ポインタのさしているページの R ビットを調べる．R=0 ならそれを追出す．R=1 なら，これを R=0 に変えて，ポインタを次のページに進め同じことを繰返す．

(d) 追出すべきページが決ったら，そのページの C ビットを調べる．C=0 なら変更がなかったしるしであるから，ドラム上にすでに同じページのコピーがあれば書き出さない．なければ，書き出す．C=1 なら常にドラムに書き出す．

(e) 新しく，追出し可能なページが1ページ増えるときには，ポインタのさすページとリスト上の後方向のページとの間に加える．

以上 FINUFO 法について説明したが，これの適用方法について次に述べる．前に説明した LRU の仮定はプログラムの局所性という性質にもとづいて立てられたものである．多重プログラミングでタスク・スイッチが頻繁に行なわれるケースでは，全タスクの空間を全体的にまとめて観察すると，LRU の仮定は成り立ちにくいと思われる．そこで OS 7 では全タスクで共通に使われるシステム空間については，これ全体を1つの環状リストにして取扱い，各ユーザ空間についてはユーザ・タスク毎に1つの環状リストにして FINUFO 法を適用している．

次に各環状リスト間の選択について説明する．

(1) 先ず，システム空間か，ユーザ空間かを選択する．これは両空間に割当てられたページング可能なページ，1ページ当りのマッピング・フォールト発生回数の少ない方を選択する[4]．

(2) ユーザ空間が選択されたときは，さらにどのユーザ・タスクの空間からページを取り上げるかを決定する．メモリを大量に使う特別なジョブが同時に走る他のジョブから不当にメモリを奪い，システム全体に混乱を起したり，スループットを低下させたりすることを避けるために，ユーザ・タスク間の選択については次のような考慮が払われている．各ジョブ (ユーザ・タスク) に主記憶枠 (保証される主記憶のページ数) を設定し，これを越えたタスクから優先的にページが奪われる．越えたものがなければ，長く参照されないページを持つタスクが選択される．主記憶枠はジョブ・クラス毎に上限値がシステム・ジェネレーション時に決められ，その範囲内でユーザが宣言できる．

以上「追出しページの選択アルゴリズム」を説明したが，同時に実行されるタスクの多重度が大きすぎるとスワッピング回数が非常に多くなり有効な仕事ができなくなる．この現象を Thrashing と呼んでいるがこの状態になったことを感知すると，管理プログラムは適当なタスクを選び一時的にこれをインアクティブな状態にし，実効的な多重度を減らす制御を行なう．

2.2 多重プロセシング

2.2.1 ロック方式

OS 7 管理プログラムは割込み禁止で実行される部分と割込み許可で実行される部分とがある．前者はその性質上，実行中にマッピング・フォールトも発生しないように制御しなければならないので主記憶常駐を減らすためには，できるだけ多くの部分を後者の構造にしておく方が望ましい．後者の部分はシングル・プロセッサでも割込みを契機に任意の場所でタスク・ス

イッチが起こり，同じ部分を他のタスクで実行する可能性があるため，論理的にはマルチ・プロセッサでも，シングル・プロセッサでも同じ構造でよい．したがって多重プロセシングの最も簡単な実現方法は前者の部分を各 CPU 間でシリアルにしか実行しないように制御することである．これは割込み禁止で実行する管理プログラムを1つのシリアルに使用可能な資源とみなしてロックしていることになる．一般にプログラムの手続部分は自分自身の書き替えを行なわなければ CPU 間でパラレルに使用可能であるからデータ部分のみをシリアルに使用可能な資源とし，複数個の資源に分割できる．この方が CPU 間の資源の奪い合いによるロス時間を短縮できる．OS 7 では数十個の資源に分割し，マルチ・プロセッサの資源の奪い合いによるロスを最小にしている．また複数資源に対するロックによるデッドロックは定順にかけることにより回避している．ロックの方法は2種類あり，TS (Test and Set) 命令によってロックし，不成功の場合再び TS 命令を繰返すスピン型と，不成功の場合 CPU を他のプロセス†に明け渡す CPU 譲渡型がある．後者は割込み許可の部分でも使用できる．

2.2.2 タスク・スケジューリング

OS 7 では2種類のスケジューリング・アルゴリズムをシステム・ジェネレーション時に選択できる．1つは単純なタスクの優先度にもとづくプライオリティ・スケジューリング，他の1つはシステム・タスク，実時間タスク，会話タスクおよびバッチ・タスクの各タスク種別毎にキューを持つスケジューリングである．

同一タスク種別中では優先度順にスケジュールされ，各種別間では普通は上記の順でスケジュールされる．会話とバッチの CPU 使用率による制御を行なう場合には，システム・ジェネレーション時に与えた比率以上に会話タスクが CPU を使ったときは，スケジューリングの順序が逆転する．また後で述べる 8700 と 8800 の混合システムでは 8800 についてはシステム・タスクとユーザ・タスクの順序が逆転する．

いずれのスケジューリング方法でも n CPU に対して1組のレディ・タスク・キューがあり，先頭の n タスクが選択されて実行される．マルチ・プロセッサでは次の場合にダイレクト・コントロールを用いて他 CPU へ連絡する．

（1） レディ・タスクができたとき，自 CPU 以外でアイドル状態の CPU があった．

（2） レディ・タスクができたとき，自 CPU 以外でそのタスクより優先度の低いタスクを実行中の CPU があった．

連絡を受けた CPU は再度スケジューリングをやり直す．

2.2.3 異種プロセッサによる多重プロセシング

OS 7 においては 8800 と 8700 の混合マルチプロセッサが可能である．混合マルチプロセッサには次の利点がある．

（1） システムを構成する場合の CPU 処理能力の選択を広い範囲で行なうことができる．

（2） CPU の機種によってそれぞれ得意な仕事を分担させることにより，システムの処理能力を向上させることができる．

混合マルチプロセッサのために OS 7 では次のことを行なっている．

（1） システム・タスクは 8700 で，またユーザ・タスクは 8800 で優先的にスケジューリングされる．（すなわち 8700 では通常の順でシステム・タスク→ユーザ・タスク，8800 ではユーザ・タスク→システム・タスクの順に選択する．）

これは CPU の機種による特長を生かすためであり 8700 と 8800 を比較すると 8800 は相対的にユーザ・プログラムの処理が得意である．

（2） 8800 は 8700 より先にアイドル状態にならないようにしている．8800 がアイドルになるときに 8700 が何か仕事をしていれば，これを取り上げて 8800 がやり 8700 がアイドルする．

（3） 入出力制御をすべて 8700 に受持たせることができる．（一般にシステム・ジェネレーション時にチャネル対応に入出力命令を発行し，その終了割込みを受付ける CPU を指定できる．）

これにより 8800 は入出力要求をチャネルのキューに登録するだけでよい．チャネルが動作中であれば，終了割込み発生時に 8700 がキューからその要求を取り出し実行する．チャネルが動作中でなければ，8800 から 8700 にキューに要求が登録されたことを連絡する．

（4） CPU のスピードが異なるため，ユーザ・タスクの使用した CPU 時間は一定の換算率により 8800 の時間に換算される．

（5） 最も短いターン・アラウンド時間を要求する処理には「緊急タスク」を利用でき，これは最高

† タスクとアクションの総称．**3. OS 7 の構造**参照．

の優先度で 8800 でのみ実行される.

2.2.4 多重プロセシングの性能

マルチプロセッサ・システムの性能をシングルプロセッサ・システムの性能と比較すると次のような要因が考えられる.

(1) プラス要因

(a) システムの資源の分割損がなくなり，資源割当てに自由度が増す結果，資源の使用率が高くなり，トータル・スループットが増大する.

(b) プログラムの共用の効果が一層発揮され，主記憶の節約ができるため，スワッピングのオーバーヘッドが減少する.

(2) マイナス要因

(a) 主記憶を共用することによって生ずるハードウェア的なメモリ・アクセスのぶつかり合い. 8700/8800 は CPU ごとにバッファ・メモリを持つため従来のシステムより影響が小さい.

(b) シリアルに使用可能な資源の競合によるロックのぶつかり合い. OS 7 では資源を多数に分割してぶつかり合いを減少させている.

これらの要因はユーザ・プログラムの性質，環境条件によって大きな影響を受け一般的な性能評価は難しいが，OS 7 で 20 種類の FORTRAN プログラムのコンパイル & ゴーの実測を行なった結果は Table 1 のとおりである.

Table 1 Throughput of multi-processing on OS 7 (Measured on 20 FORTRAN programs)

No.	Configuration	Thrupt Ratio	CPU Time Ratio
1	8,700×1, 1 MB Memory	**1 (0.8)**	1
2	8,700×1, 2 MB Memory	1.2 **(1)**	1
3	8,700×2, 2 MB Memory	**2.3 (1.84)**	1.1

2.3 プログラムの共用

2.3.1 リエントラント・プログラム

電子計算機が大型になり，多重プログラミング，多重プロセシングを本格的に行なうようになるにつれて，システム内で複数のユーザが同時に同一のプログラムを使用する機会が多くなって来た. 従来の計算機システムでは管理プログラムのごく一部分のみがメモリ中に 1 コピーのみ置かれて共用され，コンパイラ等は各ユーザ毎にメモリ上にコピーを持ち共用されないものが多かった. OS 7 では管理プログラムはもちろんのこと，コンパイラ，ユーティリティ，各種ライブラリ等の全部のシステム・プログラムがシステム空間に置かれ，メモリ上で 1 コピーで全ユーザから共用できる. ユーザの作成したプログラムもリエントラントになっていれば，システム・プログラムと同様の方法で，共用させることができる.

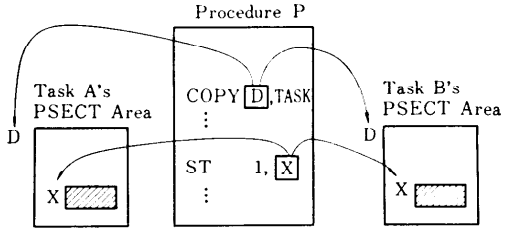

Fig. 4 Reentrant program using PSECT

OS 7 では処理プログラムをリエントラントにする手段として PSECT (Private Section) を利用することができる. プログラム作成時に PSECT 中にデータ領域を定義しておき，実行時に COPY マクロを発行して，タスク毎に別のデータ領域を取ることができる. Fig. 4 はその使用例である.

FORTRAN, COBOL, PL/I 等のコンパイラを用いた場合にはユーザの指定により，リエントラントなオブジェクト・プログラムを作ることができる. この場合ユーザは PSECT を意識する必要はない. コンパイラが全部面倒をみてくれる.

PSECT の他にリエントラント・プログラムを作る有効な手段がもう一つある. サブルーチンの呼出しのときタスク毎にコントロール・スタックとデータ・スタックと呼ばれる 2 種類のプッシュ・ダウン・スタックを用意している. これはハードウェアの用意した専用の 5 種類の命令 (SKB, SKC, RTN, GSK, FSK) によって利用できる. 特に管理プログラムは PSECT を使用できないので，コントロール・スタックを有効に利用してリエントラントなプログラムを作成している.

2.3.2 論理オーバーレイ構造

従来の計算機システムでコンパイラ等をジョブ間で共用することができなかった理由は大抵これらはオーバーレイ構造になっており，Fig. 5 のように各フェーズが主記憶上の同一アドレスにローディングされるた

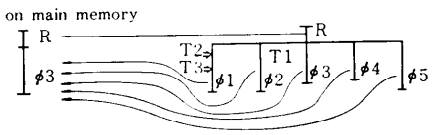

Fig. 5 Overlay structure in real memory system

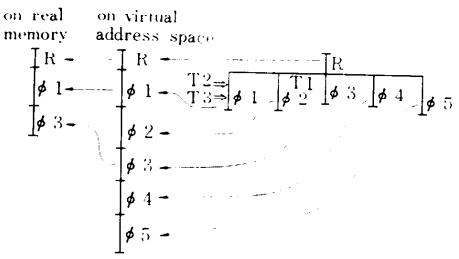

Fig. 6 Logical overlay structure in virtual memory system

め，φ1～φ5のうち唯1つしか主記憶上に置くことができず，共用した場合，あるタスク T1 が φ3 を使っていると他のタスク T2, T3 は φ1 を使うことができない．これに対して OS 7 では論理オーバーレイ構造という概念を導入し，この問題を解決している．

Fig. 6 のように各フェーズはロード・モジュールという単位にまとめられ，仮想アドレス空間上では別のアドレス上に置かれるため，φ1 と φ3 が同時に仮想記憶および主記憶上に存在し得るからである．次に論理オーバーレイ構造の制御方式について Fig. 6 に従って説明する．

タスク T1 が φ3 の処理を終了し，φ4 にリンクするとリンク・フォールトが発生し，管理プログラムは「T1 は φ4 を使用せんとしているので，これと排反の関係にある φ3 はもはや必要としなくなった．また他に φ3 を使用中のタスクもないので，φ3 の占有していた主記憶は不要になった」ことを知る．

このように管理プログラムは常にどのロード・モジュールは何個のタスクで使用されているかを管理しており，使うタスクがなくなったとき，そのロード・モジュールは主記憶から追出される．これをデマンド・ページングと比較すると，デマンド・ページングは前に説明したように追出しページを FINUFO 法等で見つける訳であるが，これは「近い将来使われないであろう」という推測にすぎない．これに対して論理オーバーレイは追出しページが積極的に管理プログラムに知らされることになり，プログラムの動きに従った正確な情報をもとに制御しているので，うまく使えば効率がよい．

2.3.3 ダイナミック・リンク

複数のプログラム・モジュールを実行に先立ってリンケージ・エディタによって結合し，単一のプログラム・モジュールにして実行させるスタティック・リンクの他に OS 7 では実行時に結合するダイナミック・リンク機能がある．

このため言語プロセッサの出力のオブジェクト・プログラムの形式とリンケージ・エディタの出力形式は同一であり，したがって言語プロセッサの出力はそのままで実行可能である．

この機能を利用してライブラリ・サブルーチンはスタティック・リンクによってユーザのプログラム・モジュール中に取り込むこともできるし，ダイナミック・リンクによって実行時に呼び出し，他のジョブと共用することもできる．

2.4 コマンド・システム

OS 7 では人間が計算機システムに与える指令としては次のようなものがある．

（1） ユーザがバッチ・ジョブに関する指令として与えるジョブ制御言語 (JCL).

（2） ユーザが TSS 端末から与えるユーザ・コマンド.

（3） オペレータがコンソールから与えるオペレータ・コマンド.

（4） 端末オペレータがリモート・バッチの端末から与えるリモート・バッチ・オペレータ・コマンド.

これら(1)～(4)はすべて統一的に“コマンド”として取扱われている．特に (1), (2) は総称してユーザ・コマンドと呼ばれ，同一コマンドが原則としてどちらの形態ででも同じように使用できる．したがってユーザは一種類のコマンドを覚えるだけでバッチ，TSS の両形態を使いこなせることになり便利であると同時に，両形態間でプログラムの共用が行なわれることになり経済的でもある．

前述のコマンドはすべて同一のコマンド・アナライザ・モジュールで文法上の解釈，チェックが行なわれる．

一般にコマンドはユーザ・コマンドにせよオペレータ・コマンドにせよ各計算機センタの運用方法，各ユーザの好み等により，変更，追加を行ないたいことがよくある．特に TSS を利用するときにはユーザは自分に必要なコマンドを作成したり，既存のコマンドの仕様を自分の好みに合せて一部変更して使い易く，覚え易いものに修正したりしたくなるものである．

このような要求を満すため，OS 7 のコマンド・システムは「コマンドの定義」を「コマンド・アナライザ」から独立させ，ライブラリの形式で与えることを許している．これを「コマンド・ライブラリ」と呼ん

でいる.

さらに，コマンドの部分的な修正——コマンド名，キー・ワード名に対する同義語の設定，オペランドに対する標準値の変更——を簡単に行なえるようにするため，コマンド・ライブラリとは別に修正部分のみを記録する「プロファイル」と呼ばれるファイルを持っている.「コマンド・ライブラリ」と「プロファイル」はシステムの持っているシステム・コマンド・ライブラリ，システム・プロファイルの他に各ユーザが自分専用のユーザ・コマンド・ライブラリ，ユーザ・プロファイルを持つことができる．専用のものを持っていないユーザは，システムの持つ標準のコマンド体系のみが使えることになる.

ユーザのジョブの開始時にコマンド・ライブラリとプロファイルからコマンドの解析に必要な全情報を集めた結合辞書とよばれるものが仮想記憶上に作られ，コマンド・アナライザによって参照される．コマンド・ライブラリ上に定義できるコマンドには組込みコマンドとマクロ・コマンドの2種類がある.

組込みコマンドはコマンド名とコマンド処理ルーチン名とオペランドのキー・ワード名およびオペランドに関する規則をコマンド・ライブラリ上に定義する.

マクロ・コマンドはアセンブラ・マクロの定義と同様の方法で，既存のコマンドを組合せて，新たに1個のコマンドを定義する．コマンド・ライブラリはユーティリティを用いて簡単に作成，変更ができる.

プロファイル中には次のものが格納されている.

(1) 標準値
コマンドのオペランドを省略した場合とられる値を標準値と呼ぶ．DEFAULT コマンドによって変更することができる.

(2) 同義語
SYNONYM コマンドでコマンド名，オペランドのキー・ワード名に新しい名をつけ，以後これを用いることができる.

(3) コマンド記号
管理プログラム，ユーティリティとユーザとのインタフェースになる変数で SET コマンドで変更できる.

以上述べた DEFAULT, SYNONYM, SET コマンドで変更されるのは結合辞書中の値であり，これらをプロファイル中の値に反映するには PROFILE コマンドを使用する．コマンド・ライブラリとプロファイルと結合辞書の関係を Fig. 7 に示す.

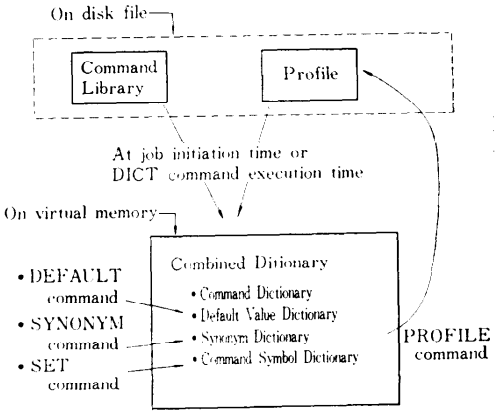

Fig. 7 Relationship between command library, profile and combined dictionary

3. OS 7 の構造

OS 7 には処理の単位としてタスクの他にアクションと呼ばれるものがある.

アクションはいわば，簡易タスクとでもいうべきもので，一般には外部割込み発生時に作られ独立のコントロールの流れを持ち外部割込みに対する処理をすべて終了すれば消滅する．特別な処理のために，ソフトウェア的要因でアクションを作ることもある．メモリ・スケジューリング，チャネル・スケジューリング，入出力エラー回復等はすべてアクションで処理されている．アクションは常にタスクよりも優先して実行される.

タスクが別個のユーザ空間を持つのに対し，アクションは外部割込み発生時にたまたまランしていたタスクの空間を借用して実行される．したがってアクションは一般にユーザ空間を参照しても意味がないのでシステム空間のみを参照する.

アクションは割込みを許可した状態で実行でき，マッピング・フォールトを起してもよい．このときタスクより優先してページの割当てが行なわれる．したがって割込み（マッピング・フォールト割込みは除く）によって起動される処理ルーチン自身およびそれが参照する領域をページング可能にできるので，アクションの利用により主記憶常駐ページを減少させることができる.

また，独立の処理の単位が増えることになり，CPU の使用効率を上げることができる．特にマルチプロセッサ・システムではその効果が大きい.

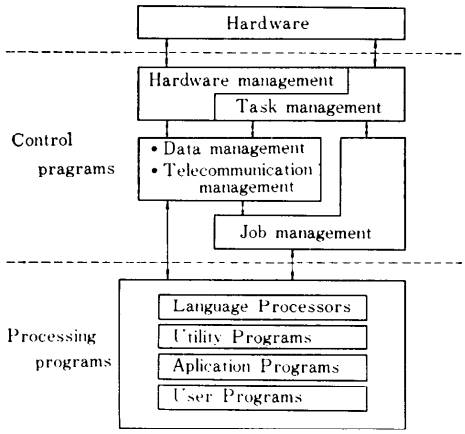

Fig. 8 Structure of OS 7 system

また，タスクにはシステム・タスクとユーザ・タスクがあり，すべてのシステム・タスクは同一の仮想空間で動作する．独立のジョブを形成する各ユーザ・タスクは別個の仮想空間を持っている．システム・タスクはユーザ・タスクより優先して処理される．

OS 7 管理プログラムは，処理単位からみると次の3つの部分に分類される．

（1） アクションで処理される部分．

（2） タスク（システム・タスクまたはユーザ・タスク）の延長でルーチンとして処理される部分．

（3） 独立のシステム・タスクで処理される部分．

次に OS 7 の構造を Fig. 8 に示す．

（1） ハードウェア管理

ハードウェアの入出力インタフェースの仕様上の差を吸収し，仮想アドレスで表現された入出力の論理コマンドがハードウェアの仕様に合う実アドレスの物理コマンドに変換され実行される．ハードウェアの割込み信号をアクションに変換するのもこの部分である．タスクの延長で処理される部分とアクションで処理される部分がある．

（2） タスク管理

CPU 資源，メモリ資源，プログラム・ライブラリ資源の管理を行なう．タスクという処理の単位を生成し，起動，停止させるのもこの部分である．タスクの延長で処理される部分とアクションで処理される部分がある．

（3） 通信管理

通信回線，端末との入出力をコントロールする．タスクの延長で処理される部分とアクションで処理される部分と独立のシステム・タスクで処理される部分がある．

（4） データ管理

各種ファイルに対するアクセス法，ファイル・スペースの管理，カタログの管理を行なう．すべてタスクの延長で処理される．

（5） ジョブ管理

ジョブの入力と出力，コマンドの解析，ジョブのスケジューリング，装置の割当て，コンソールの制御，TSS ジョブの制御およびリモート・バッチの制御を行なう．タスクの延長で処理される部分と独立のシステム・タスクで処理される部分とがある．

（6） 処理プログラム

言語プロセッサ，ユーティリティ・プログラム，アプリケーション・プログラム，ユーザ・プログラムが含まれる．すべて独立のユーザ・タスクで処理される．

4. むすび

OS 7 は昭和47年10月に東京工業大学情報処理センターでシングル・プロセッサ・システムが稼動開始し，昭和48年1月には東京大学大型計算機センターでマルチプロセッサ・システムが稼動開始した．同センターでは 8800 2台と 8700 2台の 4CPU のマルチプロセッサ・システムでオープン・バッチ，クローズ・バッチ，TSS およびリモート・バッチが使用されている[2)3)]．

最後に，OS 7 の設計に当たってユーザの立場からいろいろ御助言，御意見をいただいた東京工業大学および東京大学の方々に深く感謝します．

参考文献

1) 益田，高橋，吉沢：ページング・マシンにおけるスワッピング・アルゴリズムの比較とプログラムの動作解析，情報処理，Vol. 13, No. 2, pp. 81～88 (1972).

2) 高橋，石田：東大超大型計算機システムの構成と特徴，電子通信学会電子計算機研究会資料 (1973年5月).

3) 石田：東大超大型計算機システムにおける運用上の省力化について，電子通信学会電子計算機研究会資料 (1973年5月).

4) 大西，秋田，野口，池田：HITAC 8800/8700 オペレーティング・システム (OS7) の特徴，電子通信学会電子計算機研究会資料 (1973年5月).

5) 野口，萩原，井上：OS 7 の多重プロセシングについて，電子通信学会電子計算機研究会資料 (1973 年 5 月)．
6) Denning, P. J.: The working set model for program behavior, Comm. of ACM, Vol. 11, No. 5, pp. 323～333 (May 1968).
7) Corbato, F. J.: A paging experiment with the Multics system, In Honor of Philip Morse, MIT Press (1969).
8) Denning P. J.: Virtual Memory, Computing Surveys, Vol. 2, No. 3, pp. 153～189 (September 1970).
9) Witt, B. I.: M 65 MP: an experiment in OS/360 multiprocessing, Proc. 1968 ACM National Conf., pp. 691～703 (1968).
10) IBM: OS/VS 1 Planning and Use Guide, GC 24-5090-1, File No. S. 370-34 (1973).
11) 堂免，高須，日置，長谷川：HITAC 8700 オペレーティング・システム，日立評論，Vol. 54 (1972)．
12) 日立製作所：OS 7 管理プログラム解説 (プログラム・マニュアル)，(1971)．

(昭和 48 年 7 月 6 日受付)