E7　グループ管理用モニタ

石田晴久，長谷部紀元，野本征子（東京大学大型計算機センター）

1. はじめに

電子計算機のOS（オペレーティング・システム）は近年その機能の拡大および汎用化とともに巨大化の一途をたどっている．たとえばコア128キロ語のうちOS（リモート・バッチ機能を含む）の平均駐在領域が68キロ語を占め，ユーザ・エリアはわずか60キロ語しかとれないシステムも出現している．さらに汎用超大型システム（多重プロセッサ，TSSを含む）ではOSの平均駐在量が175キロ語に及ぶものさえ出現しようとしている．

こうした巨大なOSになると管理プログラム部分だけで100人以上のプログラマが投入されることも稀ではない．このためプログラマ1人当りの生産性はきわめて低く，1人1日平均3ステップ程度といわれている．このことは信頼できるOSを早く安く作ることがいかに困難になっているかを物語る．

これに対する対策としては，OSをハイラーキーをもつレベル構造にし*，モジュール化を徹底することが提案されている〔1，2〕．レベル構造の一例としては，図1に示すようなDijkstraのTHEシステムがある．これはミクロにみた場合のOSの構造の例である．また高橋・亀田は図2に示すようなユーザー・モニタ（ユーザー毎に異なりうる仮想OS）を考察している．

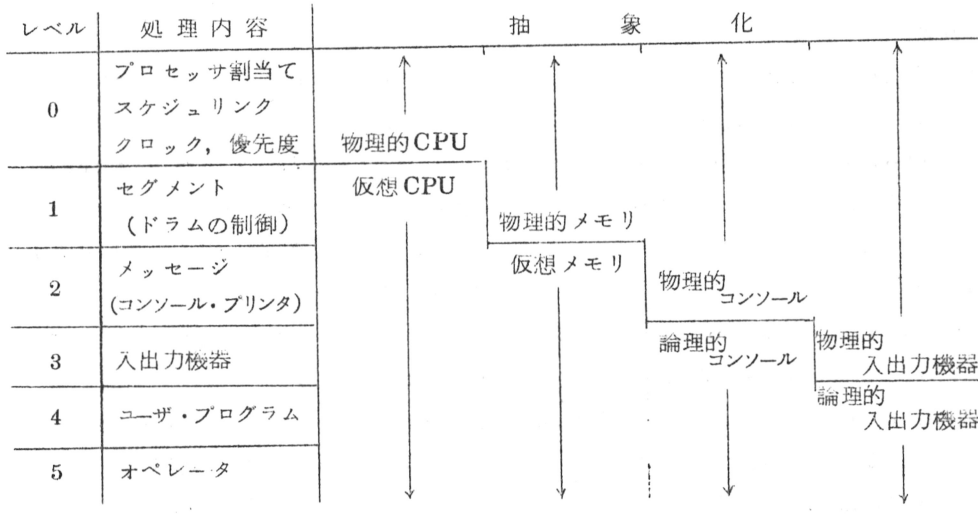

図1　THEシステムのレベル構造

* 現在のOSではシステムとユーザーの2レベルしか考えられていない．

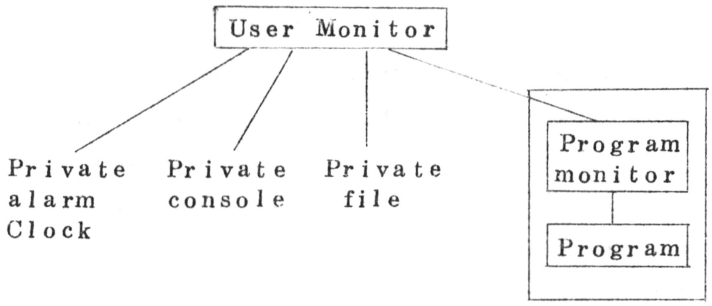

図2　ユーザー・モニタ（仮想OS）

ユーザーからみた現在のOSの問題点は汎用性がうたわれていて，たしかにいろいろな機能が織込まれている反面，ユーザーの特定の目的には必ずしも向いていない，つまり柔軟性に欠けることである．理想的には，OSが相互にある程度独立な（connectivityの低い）モジュールに分れていて，ユーザーがそれらのモジュールを適当に調合して自分の好みのOSを作成できることが望ましい．つまりOS＝基本（共通）モニタ＋オーダー・メード部分という形のdo-it-yourself monitorができるとよい．このことは図3のようにMITのMULTICSではある程度考えられていたようである．

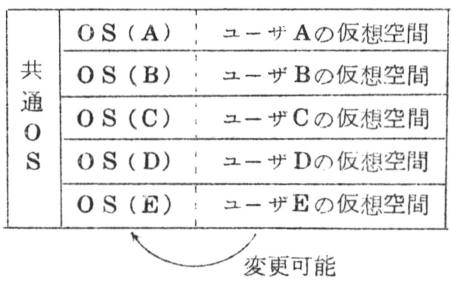

図3　仮想空間に取込まれたOS

本稿では，そのようなモニタをグループ管理モニタという形で考察し具体例をあげることにする．これはOSの機能拡大を巨大化ではなく機能の分散化ではかろうとするひとつの試みでもある．なおここではプロセスといったミクロなレベルでの考察は省くことにした．

2. グループ管理の必要性

多数のユーザーが共同で利用する大型システムでは，ユーザーあるいはプログラムのグループを管理する必要はきわめて多い．たとえば学生実習，応用プログラム開発チームなどの管理がそうである．またプログラム・グループの管理はパーフォーマンス測定のためのデータ取得とも密接に関連する．

ところが現在のOSにはグループ管理の機能は余り入っていない．たとえばHITAC8700では，(1)ファイルの3レベル構造（システム管理者・グループ管理者・ユーザー），(2)グループ内でのファイル共用（//.PERMIT *GALLなど），(3)個々のユーザーのグループ管理者の確認，(4)グループ・アカウントの集計などがあるにすぎない．ただし，グループ・アカウントの集計などの機能は実習用に作られたアセンブラに組込まれた例があり〔3〕，WATFOR，TUFFT〔4〕などのFORTRANコンパイラーへの組込みも容易であろう．

3. グループ管理モニタの機能

次に主にバッチ処理用大型システムの図4に示すような位置にあるグループ管理モニタにもたせるべき機能についてのべる．一般にはレベル数は図のように3でなく，もっと多階層でかまわない．

(1) コア常駐ではなく，普通はディスクやドラムにあって必要なときのみ呼出されるようになっていること．

(2) 厄介なシステム・ジェネレーションの手続きをしないでも，モジュールの組合せやさしかえ，あるいは既製OSのパッチ（つぎあて）で作成できること．またシステム内で共通に使える変数名（%ABCなど）を設けることも必要である．こうしたOSのファームウエア化*が可能なためにはモジュールのインターフェースの明確化がとくに重要である．またパッチをしたら使用後にその回復の手段が必要となる．なおリエントラント（再入可能）な純手続きの部分にはもちろんパッチはできない．

OSの変更には，記憶保護や優先レベルの変更も必要であるが，最近の大型機でリング保護機構のあるものなどはこれはやりやすくなっている．

(3) ジョブ管理． ジョブの連続処理を行ない，管理者がジョブ打切り条件（可変として）などを自由に制御できるようにする．また個別アカウント，グループアカウントが自由にとれ

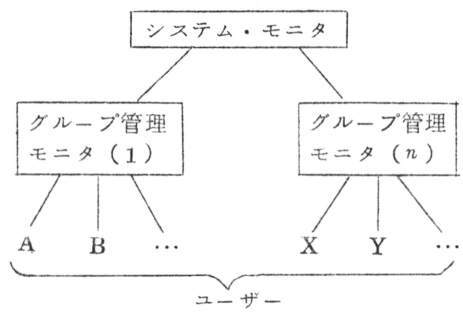

図4 グループ管理モニタ

* たとえば汎用OSの必要ないところではFORTRANモニタが簡単にできることになる．この考えは拡張可能言語によるコンパイラのファームウエア化にもつながっている．

るようにする．各ジョブの中断や条件付無視もできるようにする．これらを行なうには，アセンブラ・コンパイラ・ローダなどのシステム・プログラム自体がリロケータブル（仮想空間上でも）なサブルーチン構造をなしていることが望ましい．

最近のOSのジョブSTEPでは，ステップ毎の固定された打切り条件設定，ファイルの受渡し（PASS）や後続ステップの固定された条件（エラ・レベル）によるSKIPなどは可能であるが，それ以上の自由度はなく，ステップ毎のアカウントはとれるものもあるが，とれないものもある．

(4) タスク管理．ユーザおよび管理者レベルでのわりこみ処理*を容易にする．これが可能なためには，わりこみ処理機構の中にハイラーキーが必要である．PL／I言語では，表1のようなわりこみ（ON）条件はユーザー・レベル処理できるが，中間レベルの概念は入っていない．グループ管理では，たとえば生徒のプログラムでオーバーフローが起きたとき，先生のプログラムでそれを処理するということがあってもよいはずである．** また上位モニタへのわり出しおよび下位からのわり出しの処理も必要であろう．

表1　PL／Iにおける ON条件

AREA	FINISH	STRINGRANGE
CHECK	FIXEDOVERFLOW	SUBCRIPTRANGE
CONDITION	KEY	TRANSMIT
CONVERSION	NAME	UNDEFINEDFILE
ENDFILE	OVERFLOW	UNDERFLOW
ENDPAGE	RECORD	ZERODIVIDE
ERROR	SIZE	

(5) データ管理．とくにライブラリーや各種ファイルを使用頻度の測定機能が必要である．これはシステム・モニタで常時やっていると，オーバーヘッドが大きくなって大変であるから，必要なときにソフトウエア・モニタとしてのグループ管理モニタでやれるようにする．こうした測定データはグループ員やシステムの評価あるいはシステムのチューンアップに貴重である．

(6) ソフトウエア・システム管理．ハードウエア障害の処置を行なういわゆるシステム管理は余り必要ないが，ソフトウエア的なエラーの管理，とくにエラー・メッセージのカウントは必要である．なおできればコンパイル・エラーやローダー・エラーも管理したいが，現存の大型のFORTRANコンパイラーは5フェーズ程度に分れていてコンパイラ自体で複雑なオーバーレイが行なわれているため，エラーを捕えるためのパッチを入れることは（リエントラントでないとしても）かなり困難である．***

---

* タスク・レベルでの処理．わりこみの物理的な処理はハードウエア管理にまかされる．

** この機能は成績評価やガイダンスにも使えるであろう．

*** 管理プログラムとしては，この他通信管理（リモート入力，TSS用）がある．

(7) 柔軟な制御カードが使えること．現在のOSの制御カードでは一般にパラメータの追加，フィールドの変更（あるいはポジショナル配置の順序）などはできないが，グループ管理モニタの制御カードはその趣旨からいって管理者が自由に設計できなければならない．制御カード（コマンドあるいはマクロ）として必要なのは，たとえばグループ員の登録，個別およびグループのアカウントの初期値設定・集計・プリント，わりこみやエラーの処置などであろう．

4. グループ管理モニタの一例

以上のような考察にもとづき，われわれは基本的なグループ管理モニタを作成したので，次にそれについて報告する．

対象としたOSはHITAC 5020（32 bits／w×48 kw）のFOS（Fundamental OS）である．これはシステム・モニタ，ジョブ・モニタ，ローダからなるSPOOL（Simultaneous Peripheral Operation On-Line）型のOSである．このOSではcard-to-MT，MT-to-LPのMT5台とライブラリー用ドラムとを使って1つの実行タスク（シングル・ストリーム）と5つの入出力タスクが同時に処理できるようになっている．図5にこのFOSのメモリー・マップを示す．ユーザー・プログラム実行中にグループ管理モニタを保護するために，メモリ保護区域は変更した．

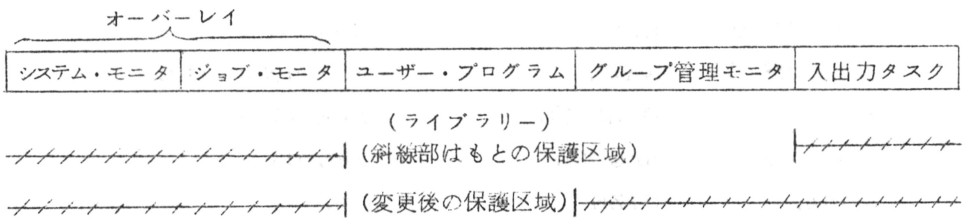

図5　FOSのメモリー・マップ

このFOSのもとでのグループ管理モニタの機能は次のような基本機能を含む．

(A) グループ員のプログラムの連続処理

(B) グループ員のジョブに対する一定の打切り条件の設定（いまのところ可変にはしていない）

(C) 個別およびグループのアカウントの集計・平均算出と印刷

(D) ある種のエラー処理

次にこうしたモニタをモジュール化されていない既存のOSに組込む方法や問題点についてのべよう．

(1) システム記述言語．当初，主要部分をFORTRANで書くことを検討した．ところがわれわれのFOSでは実行時のシステム・サブルーチンの格納場所が固定されていて，複雑なオーバーレイが行なわれているため，モニタで使おうとするシステム・サブルーチン（入出力

などの）がユーザー・ジョブ（たとえばコンパイラが同じ場所に入ってくる）でこわされてしまう．またシステム・モニタ内のメモリをひんぱんに参照する必要があるのでモニタはすべてアセンブラ語で書いた．この例のように常駐的なプログラムを高レベル言語で書くと，その言語のランタイム・システム（これを一切使わなければ問題はない）がこわされるという問題の生じる場合があるので注意を要する．

(2) OSの変更．既存OSの機能を最大限に利用することにし（図6）OSの必要部分の変

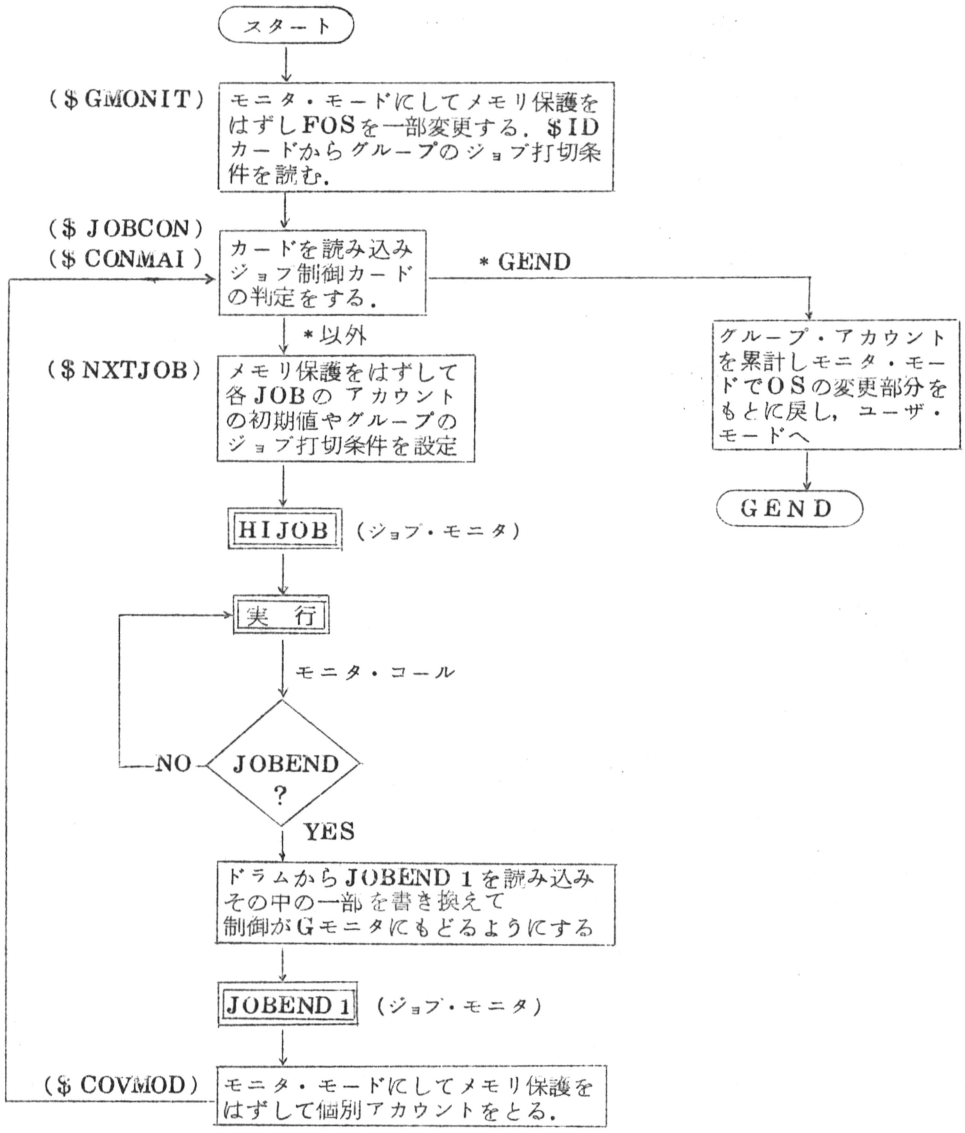

図6　グループ管理モニタ

更はパッチで行なった．これはシステムに本来あるモニタ・コールでメモリ保護を変更して行なった*．またコア常駐でない部分はドラムから呼出される都度変更を行なうことにした．

(3) 制御カード．既存の制御カードをグループ管理モニタでも読めるようにした．図7に示すように，ユーザー・プログラムは通常のランと全く同じ形になっている．最後の＊GENDはグループ管理モニタのための特別の制御カードで，これがくると（現在のところ）個別およびグループのアカウントが図8のように打出されるようになっている．

```
$              ID              (打切り条件など)
$              HIJOB           (RUNかNORUNかなど)
$GMONIT        HISAP2          (アセンブラ)
┌──────────────────┐
│ グループ管理モニタ │          (磁気テープにも入れられる)
└──────────────────┘
$              DATA            (以下はシステム・モニタのデータとして扱われる)
┌──────────────────┐
│$               HIJOB         │
│$               HISAP2        │  ユーザ・プログラム1（アセンブラ）
│                 ⋮            │
└──────────────────┘
┌──────────────────┐
│$               HIJOB         │
│$               HARP          │
│                 ⋮            │  ユーザ・プログラム2（FORTRAN）
│$               DATA          │
│                 ⋮            │
│$               EOF           │
└──────────────────┘
                ⋮
＊GEND                          (グループの終り)
$              EOF             (全ジョブの終り)
```

図7　カードの構成

(4) 既存OSの解読の困難さ．OSの中味はメーカーの機密になっているから，ユーザー側でOSに手を入れるのは大変に困難である．まずプログラム・リスト（とくに最新版の）の入手が困難であり，たとえ入手できてもコメントがなかったり不足したりしていて解読できない．メーカー側に質問しようと思っても担当者を探し出すのが大変である．とくにわれわれのFOSのシステム・モニタのように何重にもオーバーレイがしてあって，オーバレイ間のからみあいが激しい部分はほとんど解読不可能である．

そもそもユーザー側ではOSの核心部分を書きかえることはまずなく，ただかなり外周の部分を少し変更して利用したいだけである．この意味ではOSは次のように作られ，扱われるべきであろう．

＊16番地の16～23ビットにブロック上限，24～31ビットにブロック下限をセットする．1ブロックは256語．

***** GROUP ACCOUNT DATA ***** DATE 11/11/46 ID 0000FL0116

NO	NAME	START	END	USE	CPUT	COMPT	LOADT	INPUT	PUNCH	PAGE	LINE
1	S. NOMOTO	170417	170438	21	2	1	0	9	0	6	85
2	K.HIROSE	170439	170505	26	5	5	1	47	0	8	167
3	T.UASA	170506	170552	46	4	4	1	36	0	6	111
4	H.ISHIDA	170552	170616	24	4	4	1	29	0	6	121
5	S. NOMOTO	170617	170653	36	4	3	0	9	0	6	85
6	K.HIROSE	170653	170724	31	4	6	1	47	0	8	167
7	T.UASA	170725	170747	22	3	3	0	36	0	6	111
8	H.ISHIDA	170747	170809	22	3	4	0	29	0	6	121
	TOTAL			228	29	30	4	242	0	52	968
	AVERAGE			28	3	3	0	30	0	6	121

図８　グループ・アカウントの例（一部のみ）

(a) 階層構造をもつ*（たとえば図1のような）．この場合低レベルの核心部分はハードウエアの細部と同様メーカーの機密にしてよい．

(b) 高レベルの外周部分は仕様およびインターフェース（場合によってはコメント付プログラム・リスト）をユーザーに公開する．

(c) 全体としてモジュール化を徹底する．（とくに公開部分は相互の干渉がなるべく少なくなるようにする．）たとえばグループ管理モニタなどのユーザー・プログラムからひとつのモジュールとしてコンパイラが呼出せるとよい．

5. わりこみの多レベル処理機構

グループ管理モニタにぜひとも組入れたい機能に，わりこみの多レベル処理機構がある．広い意味のわりこみには，わりだし（走行中のプログラムが意識して発するもの）とわりこみ（外部要因で起こるもので，通常当のプログラムにはいつ起こるか分らないもの）がある．

このうちわりだしについては上位モニタ（上位プロセス〔2〕）をwakeupしてそれに処理をまかせることでよいであろう．問題はわりこみ（打切り時間がすぎたといったことも含む）の処理である．具体的にはOSの1ケ所でわりこみを受付けるとしても，個々のわりこみの処理は，その種類に応じていろいろなレベルのモニタで処理できることが望ましい．この場合どのわりこみをどのレベルで処理するかをどのように実際的に指定すべきかは今後の研究課題である．

参考文献

1. E.W. Dijkstra："The structure of "THE" multiprogramming system", Comm. ACM, Vol.11, No.5, pp.341～346（1968）
2. 高橋・亀田："オペレーティングシステムの1構成法"，情報処理学会誌，Vol.11, No.1, pp.20－31（1970）
3. 原田賢一："実習用アセンブラ・システム"，OSシンポジウム報告集，pp.251－265（1970）
4. 和田・山本："コア常駐FORTRANコンパイラTUFFTの作成"，東大大型計算機センター年報，No.1，pp.61－65（1971）

---

* 新しいOSには，8レベルのリング保護機構を使って，OSに0～3の4レベルの保護レベルを設けているものはある．

本 PDF ファイルは 1972 年発行の「第 13 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html