管理ルーチン*

吉村鉄太郎**

1. 管理ルーチンなぜ必要か

小型計算機だと，個々のプログラムを機械に掛けるのは，ふつうそのプログラムを書いた人たちであり，前の使用者から機械を引継いで自分のプログラムの実行に必要ないくつかの操作をして計算機をスタートさせる．もし途中で予期しない止まり方をしたり，プログラムが暴走してしまったりしたときにはコンソールに陣取ってプログラム・リストを見ながら原因を調べようとするし，それでもわからなければダンプ・プログラムを入れて記憶装置の内容をプリントしてからやっと次の人に機械を渡す．よしんば期待どおりにプログラムが動いたにせよ，実際の処理の前後に適当な操作をするため数分間計算機をあそばせなくてはならない．

計算機の使用料が比較的安かったり，正味の処理時間がたとえば数十分で，それに必要な準備が数分間ならこれでよいかもしれないが，仕事の量がふえてくるとそんなことはいっていられなくなって正味の処理時間以外のむだ時間はなるべく切りつめようとするだろう．さらに，もっと高速の計算機に代えたために同じ仕事が1分ですむようになってくると，準備のために機械の止まっている時間のほうが長くなってしまう．そこで，このような個々の仕事の処理に必要な準備作業や後始末，たとえば，プログラムのローディング，ライブラリー・プログラムのローディング，メモリー・ダンプ・ルーチンのロードと実行などをできるだけ計算機自体にやらせて，人間の介入を極度に省こうとするのが当然の成行で，これがモニタを生んだ最大の理由の一つといえるだろう．

このほか，プログラム実行中に予期しない事態（たとえば，プログラムにひそんでいた誤りのために起きた暴走，データの誤り，あるいは金物の故障）が起こってもできるだけ機械を止めずにその状態を記録し，その原因をできるだけつきとめたうえで適当なメッセージを打出して次の仕事に移るようにすれば便利である．

また最近は比較的小型の計算期でもFORTRANなどのプログラム言語を使うものがふえているが，言語プロセッサを読み込ませてから入力装置にソース・プログラムを準備し，出力されたオブジェクト・プログラムをローダとともにあらためて計算機に入れて実行させるという手続きも相当めんどうなもので，これを自動化できれば非常に都合が良い．

そのためにはプログラマが書くふつうのプログラムのほかに，前述の雑用一切を引受ける特別のプログラム（の一部）が常に主記憶装置の片隅にはいっていて，仕事の進み具合をにらんでいることが必要になる．このプログラムのことを管理プログラム（モニタ，スーパーバイザ，エクゼキュティブ・ルーチンなど呼び方はまちまちである）と呼んでいる．

良い管理プログラムがあれば計算機の能率を大きく向上することができるばかりでなく，機械の操作を比較的小数の熟練した専門家（オペレータ）にまかせることによってプログラマが時間をより有効に使うことができ，さらに操作ミスを犯す割合も著しく少なくなる利点がある．

2. 管理ルーチンはどんなことをするか

計算機のシステム・デザインが違うとモニタの構成や，機能も全く異なってしまうように見えるが，それでも必要最小限の機能というものはかなり共通している．

2.1 仕事の連続処理

前述のようにいくつかの仕事をむだ時間なく続けて処理する．たとえばFORTRANプログラムであればまずFORTRANコンパイラを呼び出してソース・プログラムをコンパイルさせ，次にローダを呼び出してコンパイルの結果できたオブジェクト・プログラムをロードさせる．そして標準ライブラリから入出力や初等関数などの基本的なサブルーチンもロードして実行を開始する．実行終了後ふたたび次のプログラムについて同じことを行なう．

このような処理をうまく行なうためにはプログラムはふつう前もって磁気テープや大容量記憶装置に入れ

* Data Processing Section, by Tetsutaro Yoshimura (Tokyo Shibaura Electric Co.)
** 東京芝浦電気（株）総務部機械計算課

ておく．そして，管理ルーチンやコンパイラ，アセンブラ，ローダおよび標準ライブラリもやはり磁気テープや大容量記憶装置に収めておき，十分短い時間で何回でも繰り返して呼び出せるようにしておかなければならない．

2.2 入出力ユニットの管理

モニタの支配下で動くプログラムは，それがプロダクション・プログラムか，またはシステム・プログラムかを問わず，入出力ユニットをそのアドレスで直接参照しないのがふつうで，その代わり特定の記号（シンボル）で呼ぶ．その記号から実際のアドレスへの変換は実行時にモニタが行なう．このような二段構えをとる理由の一つは次の例で説明できよう．たとえばアドレス A6 という磁気テープ装置が故障して使えなくなったとすると，直接 A6 を参照する命令を含んだプログラムは実行不能となる．しかし A6 と書く代わりに記号で（たとえば TAPE 6）そのユニットを表わしておき，TAPE 6 が実際にどのユニットに対応するかはモニタ内のシンボル・テーブルで決まるようにしておけば，実行時にモニタあるいはオペレータが適当なユニットを TAPE 6 に割り当てるであろうから実行には何の問題も生じない．このようなやり方を Symbolic I/O assignment などと呼んでおり，記号と実際のユニット・アドレスとの対応は動的に変えてゆくことができるのがふつうである．この考えをもっとおしすすめると，ファイルに対して磁気テープ，ドラム，ディスクなどのいくつか異なるメディアのどれを割り当てるかということまで動的に指定できることになるので，プログラムの互換性を増し，計算機をより上手に使いこなすことができるようになるだろう．ただし，このためにはファイルやレコードの論理的および物理的なフォーマットや，計算機本体から個々の入出力装置を制御するやり方（interface）などが統一のとれた規約に従っていることが必要である．

2.3 割り込みの処理

入出力と計算との並行処理，マルチ・プログラミング，実時間処理などのために割り込み機能をそなえた計算機が多くなってきた．しかし割り込みが生じたときの処置などを個々のプログラムを書くたびに考えるのは事実上ほとんど不可能であるから，きわめて一般的に書かれた割り込み処理プログラムを記憶装置内に常駐させておき，すべてのプログラムがこれを利用する方式がとられるようになった．その機能から見てこれをモニタの一部とする場合が多い．

2.4 仕事の記録

複数個の仕事を連続して処理したとき，個々の仕事にどれだけ時間を使ったかを記録しておくことは，計算システムの運用上の資料となるほか，会計操作上必要な情報でもある．モニタは各仕事の開始時と終了時にそれぞれ呼び出されるから計算機内部の時計（あるレジスタもしくは番地の内容が一定時間ごとに1ずつふえてゆくもの）を見て所要時間を記録することができる．そしてプログラム名，日時，出力量，コンパイル時間，実行時間などを適当に編集してプリントあるいはパンチする．このほか計算機によっては個々の入出力装置の占有時間まで知ることのできるものがあり，そのような情報は非常に役に立つ．

もしセットしてから任意の時間がたったあとで割り込みを起こすような時計が利用できれば，予想時間をすぎてもまだ終らない仕事を自動的に中止させ，次の仕事に移ることができる．この機能はプログラムが無限のループにはいってしまった場合などに特に有効である．

2.5 オペレータとの連絡

管理ルーチンが仕事の連続処理をするようになると個々のプログラマが機械の操作をする代わりに専門のオペレータがこれに当るようになる．オペレータのする仕事も単純なものは前述のようにモニタが代行するようになっているが，このほか仕事の進み具合，入出力装置の動作状況，誤動作の警報などオペレータがより高度の判断を下すのに役立つような情報を表示することもモニタの役目になる．

反対にオペレータが積極的にモニタに指令することができるようになっていることも重要である．前述の入出力装置の動的な割り当てをしたり，ある仕事を通常の処理順位を無視して優先的に実行させたりする場合がこれにあたる．

3. 管理ルーチンのタイプ

プログラム・テープやデータ・テープの作成，出力テープのプリント，パンチなどは小型の衛星計算機にまかせ，大型計算機はコンパイルや計算処理に専念するというオフ・ライン・システムの管理ルーチンでは，仕事の処理は原則として入力テープにはいっている順序で行なう．このようなシステムの場合はマルチ・プログラミングを積極的には行なわない場合が多く，バッファリングをつかさどる入出力制御システムで割り込み機能を利用することが多い．そしてある仕

事を処理しているときは，その仕事に関係のないプログラムは呼び込まれないので仕事の記録（アカウンティング）は比較的簡単なロジックでモニタが行なうことができる．また前に中断した仕事をふたたび続けるリランの操作も同じ理由から比較的安易である．

次にオン・ライン・システムでは管理ルーチンの考え方もだいぶ違う．オン・ライン・システムでは衛星計算機を使わず，入出力装置を直接主計算機につけてすべての入出力処理を主計算機が行なう．さらにドラムやディスクなどの大容量記憶装置をつけた場合には個々の仕事の処理順位は必ずしもそれらを計算機に入力した順序ではない．このような方式を実現するためにはまずマルチ・プログラミングを積極的に採用することが必要になる．たとえばすでに実行の終った仕事の結果(磁気テープや大容量記憶装置にはいっている)をプリントするプログラム，プロダクション・プログラム，これから処理する仕事のプログラムやデータをカードから磁気テープないし大容量記憶装置に移すプログラムなどを同時に記憶装置内に共存させ，入出力につきものの待ち時間を利用してそれらのプログラムを少しずつかわるがわる実行させてゆけば見掛け上ある程度の並行処理をすることができる．

次にこのようなシステムでは各仕事相互間の優先順位をモニタが決めることができる．すなわち，プログラムやデータは準備のでき次第入力するが，このときモニタはそのカード・イメージがディスクやドラムのどこにはいっているか，優先順位（プログラマが指定する),その処理で必要となる入出力機器や記憶容量などをしらべ，表にして持っている．そして一つの仕事が終る（すなわちそれが占有していた記憶区域，入出力機器などが解放される）とこの表を見て実行可能なプログラムのうちで優先度の最も高いものを選んでロードする．ここで実行可能という意味は記憶装置や入出力装置がそのプログラムで必要とするだけあいているということである．なお優先順位はその仕事が入力されてしまったあとでも，オペレータからの通信によって自由に変更できるようにしておく．

このようなシステムは柔軟性に富んだもので計算機の性能をよく生かすことができるが，上にあげたようなことを行なうだけでも，モニタには相当の負担がかかってくる．さらにオフ・ライン・システムでは考える必要のなかったことがマルチ・プログラミングの採用に伴なっていくつか問題になってきた．

まず記憶装置の割り付けを動的に行なわなくてはならない．たとえば1,000語を要するプログラムが実行を待っているときに，記憶装置中では250語の未使用ブロックが4個ばらばらに存在していることがありうる．1,000語の連続した区域が空くまで待つのは不経済なので，現在はいっているいくつかのプログラムやデータ区域をずらして積極的に1,000語の区域を作らなければならない．これは計算機の命令体系，モニタ，ローダなどに関係する難問題である．

次に一つの仕事の実行を中断してその状態を保存し，あとでその実行を再開するいわゆるリランもオンライン・システムのモニタにとってかなりやっかいな問題になる．マルチ・プログラミング方式をとっていると，仕事を中断したときと，後日再開したときでは記憶装置や入出力装置の割り当てが必ずしも同一ではないからである．

また個々の仕事の処理記録を行なうアカウンティング・ルーチンもだいぶ複雑になるだろう．マルチプログラミング方式では個々のプログラムがこまぎれに実行されるから所要時間がつかみにくいし，優先順位や入出力の回数まで考えに入れた機械使用料金算定ルーチンを作ろうとすると，算定規準の決め方をよく研究しなければならない．

4. その他

4.1 モニタが要求する規約

モニタの支配下で動くシステム・プログラムやプロダクション・プログラムは，モニタとの通信をスムーズに行なうため一定の規約を守って書かなくてはならない．たとえばサブルーチンを呼び出すときのレジスタの使い方は一定の規則に従わなくてはならない．このことは次のような場合に役立つ．いくつかのサブルーチンが入れ子になっているようなプログラムで誤りが起こり実行を中止してモニタに帰ったとき，モニタはどのレベルで誤りが起こったかをプログラマのために書き出してプログラマにしらせるものとする．このためにはモニタがその入れ子を逆にたどって調べ上げなければならないが，サブルーチンの呼び方をまちまちにしておいたならばこれは不可能になる．

また，すべてのシステムやオブジェクト・プログラムで呼ぶ入出力装置のアドレスは，前述の symbolic I/O assignment を行なうためにモニタで決めた標準の呼び方を採用しなければならない．このほか，プログラム中に停止命令を書かないで，そのような場合は常にモニタへのジャンプで代用することもよく見られ

る制限である.

4.2 モニタに対する批判

モニタというソフトウエアは元来計算機をより有効に利用するために考えられたものであるが，その規模が大きくなるにつれて，プログラマのほうは自分のプログラムのロジックとは本質的にあまり関係のない規約とか，コントロール・カードの使い方などを覚えなければならなくなった．さらに実行時にモニタが占有する記憶容量もばかにならないし，また現在のように機械の操作を一部のオペレータにまかせるやり方では人間と計算機の interaction を必要とするようなプログラムを掛けることは困難である．このことからも「モニタは必要悪である」といい切る人のでてくるものももっともなことである[4].

これらの点を解決するにはモニタ自体の研究開発も重要であるが，さらに計算機のシステム・デザインの進化が決定的な役割を果たすと思われる.

さらにソフトウエアの教育に当っては，モニタはオペレータのためのシステムであって，ふつうのプログラマはその存在をできるだけ意識しないですむのがその理想であるという点をはっきりさせ，FORTRAN や COBOL などと同格のシステムとしては扱わないようにすればかえってモニタの正しい理解，評価を促進するであろう.

参考文献

1) 和田英一：モニタシステム，情報処理，Vol. 3, No. 5 (1962)
2) 関根智明・鈴木正・山崎利治：モニタシステムについて，第5回プログラミングシンポジウム報告集，(1964)
3) J.M. Blatt, Ye Indiscreet Monitor, Comm. ACM Vol. 6, No. 9 (1963)
4) T.B. Steel, Jr., Operating Systems, An Introduction. DATAMATION Vol. 10, No. 5 (1964)

(昭和39年9月17日受付)