オペレーティング・システムのパフォーマンス・モニタリング*

北川　一**　萩原　宏***　金沢正憲***　藤原康雄****

Abstract

In recent years, as an operating system (OS) has increased the complexity, it has become very important to evaluate OS performance by observing its internal, dynamic behavior. Since OS performance is considerably influenced by its environment and work-load, performance monitoring at user side is necessary for an efficient use of OS. But most computing systems today do not supply with effective tools for performance monitoring.

We have been implementing the Performance Monitoring System (PMS) for FACOM 230-60 in Data Processing Center, Kyoto University. It enables observation of OS behavior with software monitoring in processing program mode without any change or generation of control program.

In this paper, the method for an internal monitoring of OS is described, and a performance evaluation by observing resource (core memory, CPU, i/o channel and device) utilization in FACOM 230-60 using PMS is shown.

1. まえがき

オペレーティング・システム (OS) の大型化，複雑化とともに，OS の作製者や利用者***** にとって，その性能や効率を測定し，設計仕様との差異，システム効率の隘路，システムの最適な構成法や使用法，システム改良の方向などを知ることの必要性，重要性が大となってきた．しかし，今日使用されている OS の多くは，そのシステム設計時にパフォーマンス・モニタリングのための機能が十分考慮されているとはいい難く，そのため実測によるシステム解析の有用性を認めつつも，手軽に，かつ効果的に実施するのがむずかしい，というのが現状であろう．

さて，京都大学計算機センター******では，OS 利用者の立場から，FACOM230-60 の OS である MONITOR-V システム[8] (以下 M-V と略す) のパフォーマンスを調査し，その有効利用を検討してきたが，従来の方法は，処理されたジョブのアカウント・データなどから，スループットと CPU，チャネル，コア・メモリなどのリソース使用効率についてその平均値的なものを算出し，それによりパフォーマンスを評価してシステム構成や運用方式の改善を計るというようなことであった．しかし，このような方法では，パフォーマンスの動的な変化を十分把握することは困難であり，また制御プログラムの動作状況（すなわち時間的および空間的オーバーヘッド部分）を知ることも容易ではない．一方，OS が定常的な状態で動作することは比較的少なく，また制御プログラムによるオーバー

---

* Performance Monitoring of Operating System, by Hajime Kitagawa (Data Processing Center, Kyoto University), Hiroshi Hagiwara, Masanori Kanazawa (Faculty of Engineering, Kyoto University) and Yasuo Fujiwara (Osaka Office, Fujitsu Co., Ltd.)

** 京都大学大型計算機センター
*** 京都大学工学部
**** 富士通株式会社大阪営業所

***** ここでは計算を行なう個々のユーザーではなくて，OS を管理・運用する計算センターなどを意味する．

****** 全国大学間の共同利用センターであり，昭和44年1月にサービスを開始した．FACOM 230-60 システム (2CPU，コア・メモリ 192k 語) が2セット（システム I，II と呼ぶ）設置されている．計算処理の大部分を占めるバッチ処理ジョブの種別と，CPU 時間およびラインプリンタ出力量の打切り制限/平均値は，急行：(1分/17秒，1500行/600行)，普通：(5分/110秒，6000行/1850行)，長時間：(20分/590秒，15000行/3480行) であり，1ジョブ当りのコア使用量およびカード入力の平均は，23k 語および 450 枚である．また，依頼および処理の件数比は，急行：普通：長時間が 10：10：1 の割合で，その大部分 (90% 以上) は FORTRAN ジョブである．

ヘッドがシステム・パフォーマンスに与える影響は大きいと考えられるので，OS 利用の最適化をさらに進めるためには，つぎのステップとして，OS 内部の振舞いを観測することが必要となる．システムを構成している種々のリソースの動的な使用状況，制御プログラムの動作によるオーバーヘッド部分，システムへの負荷とパフォーマンスの動的な関係を測定することにより，さらに有効なシステムの分析が可能となる．

ところで，OS のモニタリングを行なう場合，つぎのような点が望まれる．

（i） 測定による OS の効率低下（モニタリング・オーバーヘッドとよぶ）をできるだけ小さくする．

（ii） 測定データのロギングや処理が高速に行なえる．

（iii） 測定結果を直ちに OS へ反映できる．

（iv） モニターする内容の選択・変更・追加が容易に行なえる．

（v） OS の利用者が容易に使用できる．

しかし，これらの中には，互いに相反するような要求もあり，すべての点を満たすような方式は困難であろう．今までにも，計算機システムのモニタリングが多く行なわれてきたが，その手法についてみれば，ハードウェア・モニタリング，ソフトウェア・モニタリング，計算機複合体によるモニタリングに分けられる．

ハードウェア・モニタリングは，OS の動作にほとんど影響を与えずに測定できることが最大の利点であり，システムの部分的なところを精確に測定する場合には大変有効となるが，OS 全体の動きを総合的に把握するには，融通性，経済性の点で望ましい方式とはいえない．われわれのセンターでは，**Fig. 1** のような

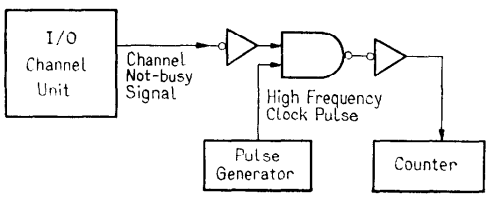


Fig. 1 Hardware Monitoring of Channel Busy-ratio.

に，計数器およびパルス発生器を簡単な論理回路によりチャネル装置に結合し，その使用率の測定を行なったが，これはハードウェア・モニタリングの最も簡単な例であろう．このような方式をさらに汎用化した測定装置も開発されている[7]．また，IBM 360 では，WAIT LAMP の点滅状況をデジタル・レコーダに録し，CPU 使用率を測定するといった例もある[2]．

ソフトウェア・モニタリングは，測定プログラムを OS の一部として組込み，制御プログラム内の種々のテーブルの内容やモジュールの動きをサンプリングおよびトレースの方法により知って，OS の動的な振舞いを観測しようとする方式であり，ハードウェア・モニタリングに比べてモニタリング・オーバーヘッドは大きくなるが，汎用性，融通性，経済性があり，OS のモニタリングにしばしば採用されている．通常は，制御プログラム内に測定用モジュールを組込むことが多いが，このような場合，制御プログラムの修正やシステム編集作業が必要となり，また測定内容の変更，追加を容易に行なうためには，作製時にかなりの汎用性，融通性を持たせておかねばならない．しかし，本論文で取り上げる FACOM 230-60 システムでは，適当な割出しの設定とその処理を行なうことにより，ユーザー・プログラムから制御プログラム領域の参照が可能である．この方法によれば，処理プログラム・モードで OS の動作状況をかなりの程度まで観測できるし，特に OS の利用者によって容易に行なえる点が便利である．

ハードウェア・モニタリング，ソフトウェア・モニタリングの両方式の長所を生かす有効な方式として，計算機複合体によるモニタリングがある．これは，CPU 間のダイレクト・インターフェイスまたは高速チャネルや高速伝送回線で結合された別の計算機システムを計算機複合体の様式で動作させ，外側から OS 内部の振舞いを観測する方法であり，測定結果の複雑なデータ処理がモニタリング・オーバーヘッドを増加させずに行なうことができる．この方式の例としては，PDP-9 による OS/360 のモニタリングを行なった SPY[5] などがあげられる．われわれは，FACOM 270-30 による FACOM 230-60 のモニタリングについて検討を行なったが[6]，それをインプリメントするためには OS の機能追加が必要であり，まだ実現していない．

以下では，FACOM 230-60 M-V 用に作成したソフトウェア・モニタリングによる効率測定プログラム PMS (Performance Monitoring System) について，その方式ならびにそれによる FACOM 230-60 M-V の観測結果などを述べる．PMS は，処理プログラム・モードで動作する測定プログラムであり，制御プ

ログラムの修正やシステム編集を特に行なわずに使用することができ，また測定内容の変更や追加が容易にできる点が特徴的である．

2. PMS

PMS は，独立な種々の測定プログラムの集りであり，それぞれの大きさは 1k 語以内である．各測定プログラムがある時間間隔毎(時計割込みによる)に収集する情報は，主として，システム動作状況（ジョブや端末の負荷状態など）とリソース使用状況に関するデータであり，それらは磁気テープに記録されるほか，一部はコンソール・タイプライタやキャラクタ・ディスプレイに出力される．また，測定時間間隔などのモニタリング・パラメータは，コンソールよりオンラインで変更することができる．

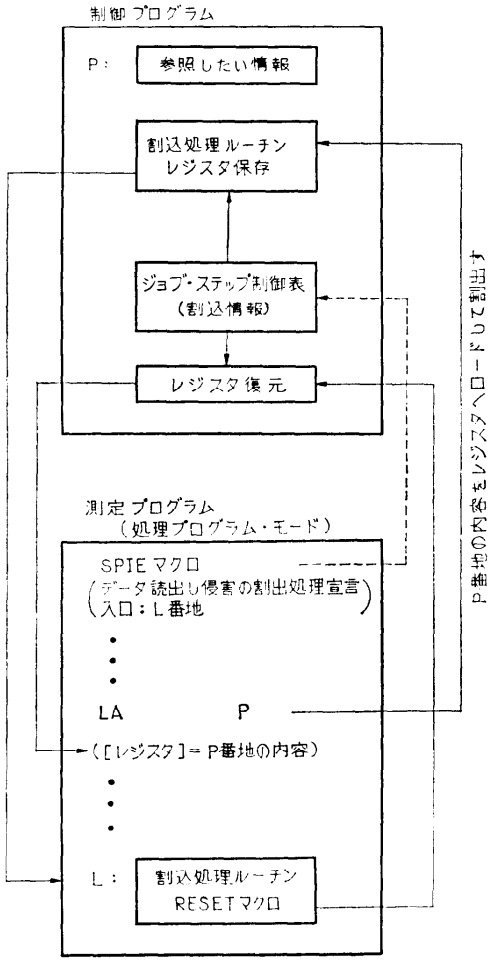


Fig. 2 Reference of Control Program Area in Processing Program Mode.

前に述べたように，測定プログラムは処理プログラム・モードであり，測定プログラムからの制御プログラム域の参照は **Fig. 2** のように行なう．すなわち，データ読出しによる記憶保護侵害の割込処理をユーザー・プログラムで行なうように SPIE (Specify Program Interrupiton Exit) マクロにより宣言して，測定プログラム領域外の P 番地の内容をレジスタへロードする命令 (LA P) を実行すれば，FACOM 230-60 では読出し侵害となる P 番地の内容をレジスタへロードしてから割出す．そして制御プログラム内の割込処理ルーチンから測定プログラムの L 番地以降にある割込処理ルーチンに制御が移る．そこで RESET マクロにより割込状態を解除すると，レジスタに P 番地の内容を残したままレジスタ・ロードの命令の直後に制御がもどる．このようにして，処理プログラムから制御プログラム領域の参照が可能となるが，レジスタへロードする 1 命令を実行するごとに割込処理ルーチンが動くため，モニタリングによる CPU 使用がモニタリング・オーバーヘッドとして問題となる．

測定時間間隔 t_m ごとに，制御プログラム内の d 語を参照するとして，1 語の参照に必要な CPU 時間を f とすれば，モニタリングによる CPU 使用率 P_M は近似的につぎのようになる．

$$P_M \fallingdotseq df/t_m. \qquad (1)$$

f は，その大部分が割込処理ルーチンの時間であり，数百 μs となるので，例えば 5 秒ごとに 100 語を参照すれば，約 1 %のオーバーヘッドとなる．実際には，P_M をさらに小さくするため，連続する 2 語の参照を 2 倍長のロード命令により行なって，割出しの回数を少なくしている．また，1 回の測定 (t_m ごとに d 語) ごとに必要な測定プログラム内の入出力時間を t_w とすると，$P_M < p$ とするには，t_m をつぎのように設定すればよい．

$$t_m > \mathrm{Max}\ (df/p,\ df + t_w). \qquad (2)$$

測定プログラムは，コア・メモリ上に単位領域である 1k 語を占有して常駐するが，後に述べるユーザー域内の空き領域の存在から考えて，コア・メモリの使用がスループットに与える影響はほとんどない．なお，測定プログラムはジョブ・ストリームを一つ占有するので，測定時は通常よりマルチプログラミングの多重度を一つだけ多くしておく必要がある．

3. コア・メモリ使用状況の測定

コア・メモリは，システムを構成しているリソースの中でも比較的高価なものであり，またマルチプログラミング処理におけるスループットはコア・メモリの容量と重要な関係にある．特に，ページング機能を持たないシステム（例えば FACOM 230-60）では，コア・メモリ上の使用している領域の間に隙ができて，むだな空き領域が生じることを避けられない．また，ユーザー域のリアロケーションを必要に応じて行なうことにより，使用領域の間に生じるむだな隙間をなくする，または小さくすることができるが，FACOM 230-60 M-V では，そのような隙間を積極的になくするためのリアロケーションは行なわれておらず，そのためにコア・メモリの使用効率が低下してパフォーマンスが制限されるような場合が起る．そこで，制御プログラム内のコア・メモリ管理表を参照しながら動的にコア・メモリ上のユーザー域の使用状況を観測し，さらに使用効率の悪い状態において，ユーザー域のリアロケーションを測定プログラムから行なえるようにした．

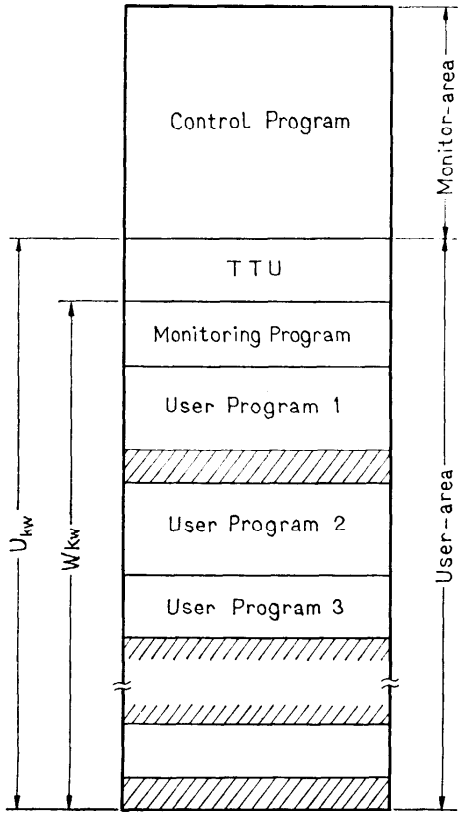


Fig. 3 Core Memory in FACOM 230-60 OS.

FACOM 230-60 M-V では，コア・メモリは **Fig. 3** のように使用されている．ユーザー域（大きさ Uk 語）は，制御プログラムの一部である TTU モジュール(ユーザー域にロードされるモジュール，システム入出力制御ルーチンなど）を含んでおり，その後にユーザー・プログラムの領域がとられる．測定プログラムは，ユーザー・プログラムの一つとして，TTU の直後に常駐させる．ユーザー・プログラムの領域の大きさはそれぞれ異なるので，ジョブ・ステップ*のイニシエート/ターミネート，ロール・イン/アウトによりプログラムが入れ替わるごとに，領域間に隙間が生じる．一方，制御プログラム内に，コア・メモリ上のユーザー域の使用状況を示すテーブル（各ジョブ・ステップへの領域割付けの様子がわかる）があり，測定プログラムは一定時間ごとにこの制御表を参照し，使用効率などを知る．コア・メモリ使用率Rは，空間的なものとして，つぎのように求める．

$$R=(1-E/U)\times 100\ (\%). \qquad (3)$$

ただし，E は，ユーザー域内の空き領域の総和 (k 語) である．

ところで，制御プログラムは，コア・メモリの有効利用をある程度考慮して，処理プログラムの領域を割付けている（要求される領域より大きい空き領域の中で最小のものを割付ける）としても，例えば，**Fig. 4**

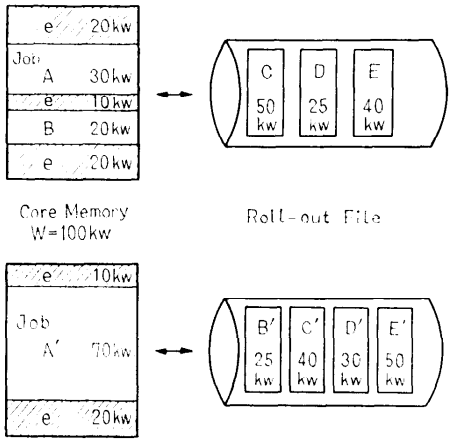
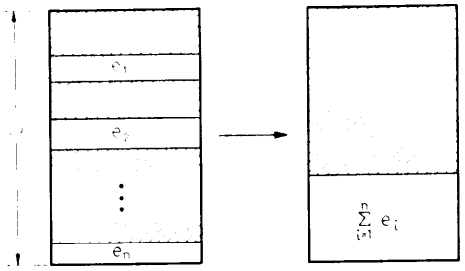


Fig. 4 Worse Cases of Core Memory Utility in a Multiprogramming System.

* 例えば FORTRAN ジョブは，コンパイル，結合編集，実行の3つのジョブ・ステップとなる．

のような隙間のある状態が起ることが十分予想される．特に，A, B, A′のジョブ・ステップが長時間で，かつ A～E, A′～E′の中で実行優先権が高ければ，このような状態は起り易く，また一度起れば，数分～十数分の間そのまま保持されることもあり得る．このとき，実効多重度は2または1となり，コア・メモリ上のジョブが入出力を行なっていることもあるので，2CPU によるマルチプログラミングの効果はなくなる．このような使用効率の悪い状態でロール・イン/アウトのきっかけが生じないことが続く場合に，ユーザー域全体のリアロケーションは，実効多重度を増加させ，マルチプログラミングのパフォーマンス向上に有効となる．しかし，一方，短時間の計算や TSS 会話型ジョブの処理が中心となるシステムでは，ロール・イン/アウトが頻繁に行なわれるので，リアロケーションを上のような目的で特に行なう必要はなく，たとえ行なったとしても，その効果がほとんどないか，またはリアローケショーのためのオーバーヘッド増加による損失の方が大きくなることもある．要するに，リアロケーションをどのような条件のもとに行なえば有効かが問題となるが，われわれは現在，つぎのように条件をきめている．

Fig. 5 のように，全体が U の大きさのユーザー域があるとき，$e_1, e_2, \cdots, e_n$ の大きさの n 個の空き領域が存在するとすれば，図のようにリアロケーション



e_i ($i=1,\cdots,n$) : Free Area

Fig. 5　Re-allocation.

するのは，

$$n \geq c_1,\ E/U \geq c_2,\ D/E \leq c_3 \qquad (4)$$

の三つの条件式が，時間 c_4 の間にわたって成立することである．しかし

$$E=\sum_{i=1}^{n} e_j,\quad D=\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_j^2-\left(\frac{E}{n}\right)^2},\qquad c_1=1,2,3,\cdots,\ 0\leq c_2, c_3\leq 1 \qquad (5)$$

であり，D は $e_1, e_2, \cdots, e_n$ の標準偏差である．このような方法により，c_1～c_4 を適当に与えれば，システムにとって無意味なリアロケーションの発生を非常に少なくすることができる．測定プログラムでは，c_1～c_4 の条件パラメータをコンソールより動的に変更可能とし，コア・メモリの使用状況をキャラクタ・ディスプレイでモニターしながら，現在の動作環境に適した条件を設定できるようにした．

一般に，リアロケーションの方法としては，

(ⅰ) コア・メモリ上の領域相互間で高速に転送を行なえる特殊なチャネル（高速転送装置）の利用，

(ⅱ) セレクタ・チャネルによりファイル経由で転送する（ロール・イン/アウト）方法，

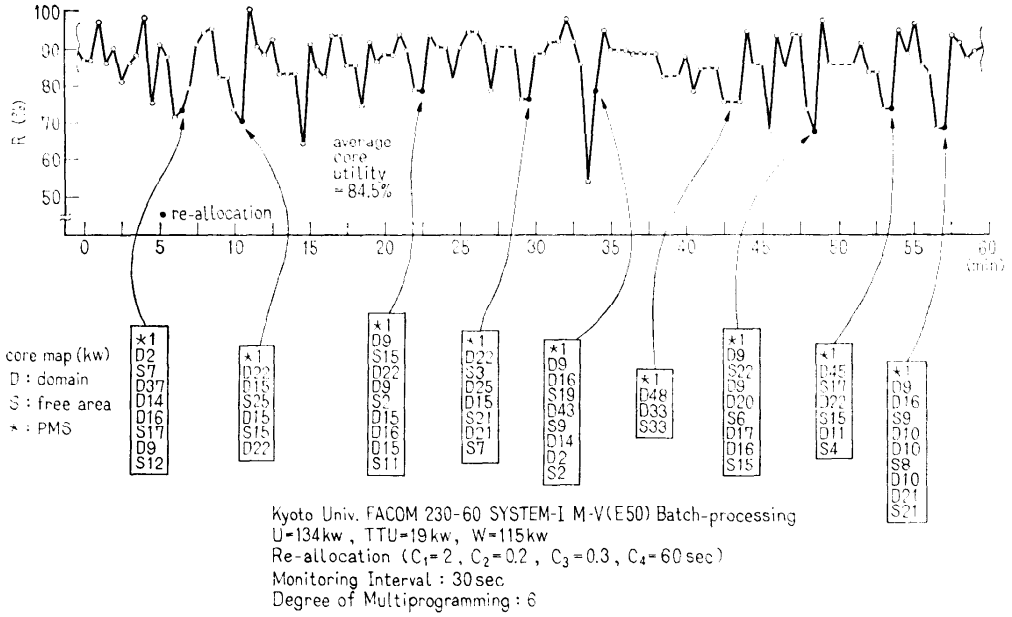
(ⅲ) CPU により直接アクセスして移し換える方法

の三つが考えられるが，リアロケーションのために生じるオーバーヘッドがシステム効率に与える影響の少ない方法を選ぶべきであろう．しかし，現在の PMS では，処理プログラム・モードであることにより，(ⅱ) の方法でつぎのように行なっている．すなわち，測定プログラムは，処理プログラム中で最高の優先度を持たせてあり，このプログラムから十分大きな領域を新しく開設するよう制御プログラムに依頼*すれば，他のプログラム（TTU を除いて）はすべてロール・アウトされる．その後に，今確保した領域を返却*すれば，ロール・アウトされた処理プログラムが空き領域をつめて連続してロール・インされ（優先度が同じプログラム間では，first-out, first-in の順），最後に連続した一つの空き領域が残る．本来，このようなリアロケーションは，ロール・イン/アウトがあまり起らない時に有効となるので，適切な条件のもとで行なっている限りにおいては，(ⅱ) の方法は悪くないと考えられるが，現在の FACOM 230-60 は，ファイル転送が比較的低速のため，リアロケーションの所要時間**に若干の問題はある．

Fig. 6 に，リアロケーションを行なった場合の，コア・メモリ使用率 R の測定例を示す．この例は，一般ジョブ処理時（バッチ処理）に約1時間測定した結果（測定間隔 30 秒）であり，平均使用率は 84.5% となっている．リアロケーションを行なわない場合の平均使用率は通常，70～80% となることが測定されており，その点で改善されている．また，$c_1=2$, $c_2=$

* 実行中のプログラムから作業域の開設およびその返却を行なうために，Get domain, Return domain というシステム・マクロ命令が用意されている．

** 1回のリアロケーションに数秒を要する．



Kyoto Univ. FACOM 230-60 SYSTEM-I M-V(E50) Batch-processing
U=134kw, TTU=19kw, W=115kw
Re-allocation (C_1=2, C_2=0.2, C_3=0.3, C_4=60 sec)
Monitoring Interval : 30 sec
Degree of Multiprogramming : 6

Fig. 6 Core Memory (User-area) Utilization Monitoring.

0.2, c_3=0.3, c_4=60 秒という条件のもとに8回のリアロケーションが行なわれており，コア・メモリ使用効率の悪い状態におけるシステムの定常化を防ぐのに役立っている．

また，**Fig. 7** には，キャラクタ・ディスプレイを使用した場合の測定結果表示例を示す．

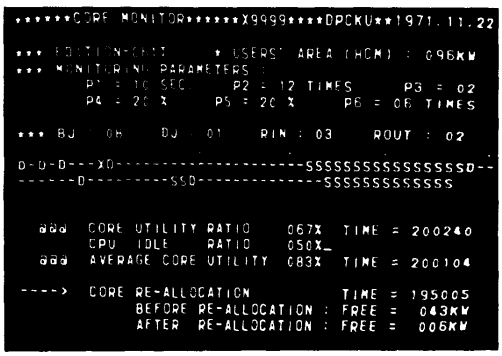
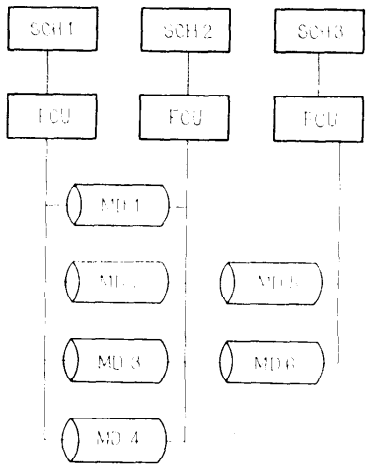


Fig. 7 Core Memory Utilization Monitoring with Character Display.

4. CPU，チャネルおよび i/o デバイスの効率測定

OS において，CPU，チャネルまたは i/o デバイスのどれかの装置が，パフォーマンスの隘路となっている場合が少なくない．この辺りの分析を有効に行なう



SCH 1~3 : Selector Channel Unit
FCU : File Control Unit
MD 1~6 : Magnetic Drum Unit
MD 1 : Control Program
MD 2 : Job Control Table/Roll out File
MD 3~5 : System Work File
MD 6 : Processing Program Library

Fig. 8 Configuration of Channel and Drum Units in Kyoto Univ. FACOM 230-60 SYSTEM-II.

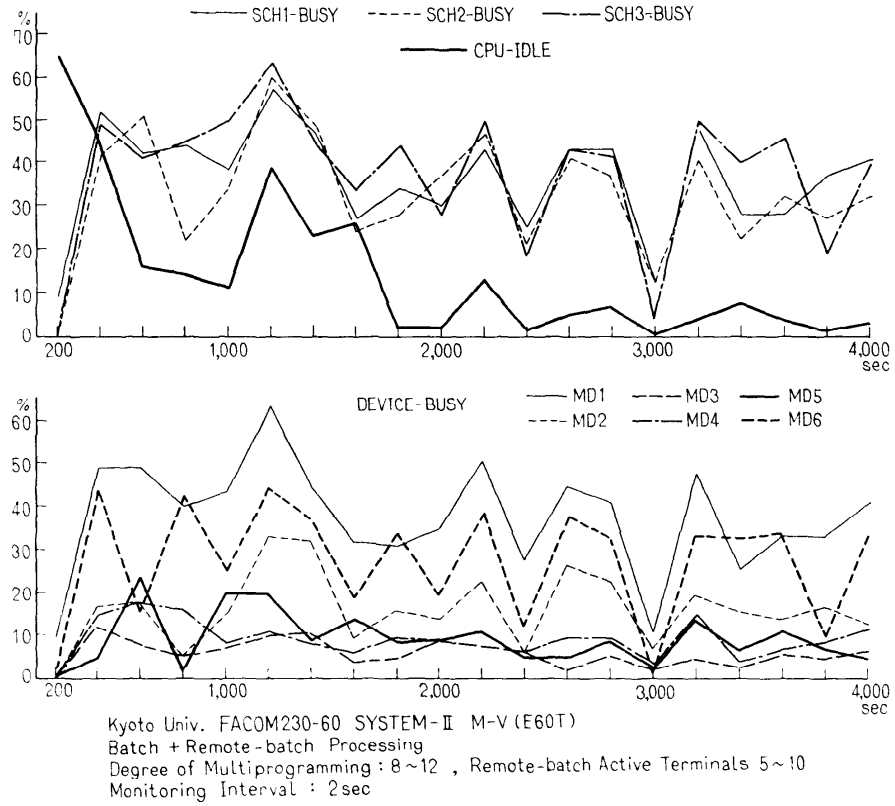


Fig. 9 Efficiency of Channel, Magnetic Drum and CPU in a Batch-processing System.

ためには，それらの装置の使用状況を並行して，かつ動的に観測することが必要となる．FACOM 230-60 M-V のような OS では，つぎの方法によりそのような観測がソフトウェア的に可能となる．

（i） 制御プログラム内にアイドル・タスクがあり，CPU 使用タスクがない場合にそのタスクが走るので，それに要した時間または回数を知れば，CPU の使用効率を測定できる．

（ii） 入出力はすべて制御プログラムの管理のもとに行なわれ，各 i/o デバイスに対する制御表に，その装置の状態や使用時におけるチャネルのパスなどの情報があるので，それらをサンプリングすることにより i/o デバイスやチャネルの使用効率を測定することができる．

PMS による測定例として，FACOM 230-60 M-V のバッチ処理と TSS 会話型処理（デマンド処理と呼ぶ）を以下に取り上げた．なお，京大センターシステムでは，システム・ファイル（制御プログラム，処理プログラム・ライブラリ，ジョブ制御表，ロール・アウト・ファイル，システム・ワーク・ファイルなど）はすべて磁気ドラム*が使用されているので，ここでは i/o デバイスとして磁気ドラムだけに注目している．すなわち，一定時間（ここでは2秒）ごとに制御プログラム内を参照し，各磁気ドラムが使用中かどうか，使用中の場合どのチャネルからのパスになっているか，CPU のアイドル・タスクの回数（1回が約 320 μs）およびジョブや端末の状況を記録して，ある時間（この例では，100 回≒200 秒）ごとに，チャネル，磁気ドラムの使用率，1 CPU 当りのアイドル率を算出する．この場合のモニタリング・オーバーヘッド P_M (CPU 使用率の測定誤差) は約 1%である．このような測定プログラムにより，2種の OS, M-V. E 60 T (バッチ＋リモート・バッチ) と M-V. E 61 T (バッチ＋リモート・バッチ＋デマンド)に関して観測

* FACOM 624k, 容量 2.56 MB, 平均アクセス・タイム 17 ms, 転送速度 0.156 MB/秒.

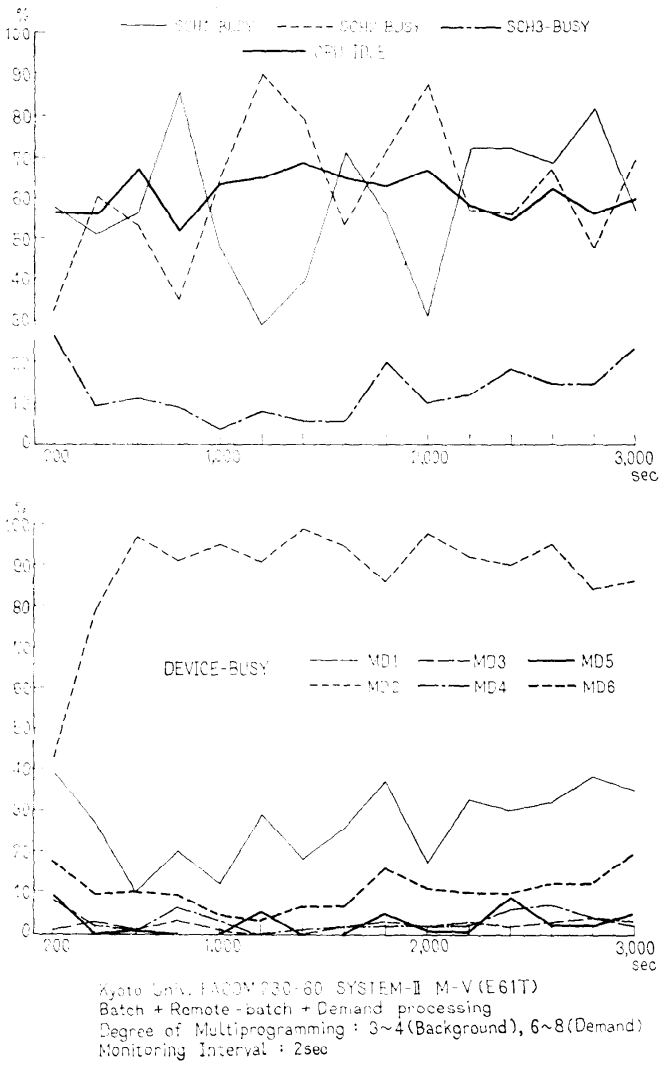


Fig. 10 Efficiency of Channel, Magnetic Drum and CPU in a Time-sharing System.

を行なった例を，**Fig. 9**，**Fig. 10** に示す．これらは，いずれも京大センターのシステムⅡを使用したものであるが，そこでのチャネル/磁気ドラムの構成と使用法は，**Fig. 8** の通りである．

Fig. 9 では，デマンド処理がないため，ロール・イン/アウトよりもむしろ制御プログラムの非常駐部分や処理プログラムのローディングに，チャネルや磁気ドラム（MD 1，MD 6）が多く使用されている．また，適切なジョブ・ミックスのもとでは，CPU のアイドル率は十分低くなっている．FACOM 230-60 のように，各リソース間で CPU が比較的高速な場合，CPU の能力を一ぱいに使用している状態が理想的であると考えられ，この例では理想に近い状態になっているものと推定される．

一方，**Fig. 10** では，デマンド処理によるロール・イン/アウトがシステム効率の隘路になっている．6～8台の端末がアクティブで，さらにバックグラウンド・ジョブとして多重度3～4のバッチ処理を行なえば，ロール・アウト用の磁気ドラム MD 2 の使用率が 90% 以上となり，その結果，CPU アイドル率が 50～60%，端末における平均応答時間が4～8秒となって，良いパフォーマンスとはいえない．この原因はつぎのように考えることができる．FACOM 230-60 M-V では，コア・メモリのユーザー域（この例では，96 k 語）を，バッチ用，デマンド用に特に分けず，各ジョブステップにおける実行優先権，コア占有優先権によりロール・イン/アウトが行なわれているが[3]，一般に，TSS による効率低下をバックグラウンド・ジョブにより回復させるため，実行優先権は，デマンド＞バッチ，コア占有優先権は，バッチ＞デマンドの如く与えるのが普通である．その結果デマンド・ジョブは端末入出力中自動的にロール・アウトされることになるが，アクティブ端末数およびバックグラウンド処理の多重度の増加とともに，デマンド・ジョブだけでなくバッチ・ジョブのロール・イン/アウトも頻繁に行なわれるようになるため，ロール・イン/アウトがシステム効率の隘路となった．なお，デマンド・ジョブへの CPU サービスはタイム・スライシングにより与えられるが，この場合タイム・カンタムは 200 ms であり，端末からの1回の要求を処理するのに十分大きな値であるので，カンタム・オーバーによりロール・アウトが増加することは，ほとんど影

響していない.

ロール・イン/アウトによるTSSの効率低下を改善するために，つぎの方法が有効である.

(1) 磁気ドラムなどのロール・アウト・ファイルの高速化，プログラム・サイズの縮少，リエントラ化の徹底により，ロール・イン/アウト時間を短縮する.

(2) バックグラウンド・ジョブの種類や多重度の制限，大容量コア・メモリ* (LCM) の使用によりロール・イン/アウトの回数を減らす. 特に，インタープリタ方式の会話型言語処理では，リエントラントなプロセッサ部分は高速のコア・メモリ (HCM) に常駐させ，ユーザーごとに必要なデータ領域を大容量コア・メモリ上に置けば，CPU の速度をほとんど落さずに，同じコストのもとでロール・イン/アウトの回数を小さくすることができる[4].

(3) システム・ファイルの割付けを変更して，チャネルや磁気ドラムの使用のバランスをとる. 例えば，Fig. 8 において，MD 2 のロール・アウト・ファイルとジョブ制御表の分離，MD 2 と MD 6 の入れ換えなどにより，制御プログラムのローディングとロール・イン/アウトを円滑化する. システム・ファイルの割付けはシステム編集時に行なう.

5. あとがき

以上の他にも，制御プログラム域内のシステム作業域や非常駐タスク・ロード域の使用状況の動的変化を観測し，その大きさの最適値や非常駐モジュールの使用頻度を求めてシステム編集作業に反映させたが，これもコア・メモリの有効利用に役立った.

汎用のシステムでは，OS の動作する環境によりその効率が異なってくる場合が多く，そのため利用者側で収集し，分析しなければならないデータもあるので，ここに述べたようなユーザー・プログラムによるモニタリングの方式は有用であると思われる. 今後の OS では，パフォーマンス・モニタリングの機能が標準的な仕様として組込まれるべきであろう.

ところで，PMS のようなパフォーマンス・モニタリングの機能が備わったとして，実際に測定を行なう場合につぎのような問題がある. すなわち，種々のシステム・パラメータの変更に対してシステム・パフォーマンスがどのように変化するかを知ろうとするとき，システムへの負荷を同じように与えることの困難さである. 特に，TSS などでは，全く同一な動作環境の再現はむずかしい. また，実時間測定であるために多くの計算機時間が必要となる. このような実測方式の問題点を解決するには，やはりシミュレーションの技法を取り入れることが有効であり，両方式の長所を生かすような，または相互にフィード・バックさせながら進めるようなシステムの分析法が望ましいと考えられる.

最後に，PMS は京都大学大型計算機センターの開発計画の一つとして行なわれたものであり，この研究開発に対して，日頃熱心にご討論していただき，またいろいろとご援助下さった京都大学大型計算機センターの星野聰氏および飯田記子氏に感謝します.

* LCM のサイクル・タイムは 6 μs (HCM は 0.92 μs)，コストは HCM の約 1/4.

参考文献

1) P. Calingaert: System Performance Evaluation: Survey and Appraisal, Comm. ACM, Vol. 10, No. 1 (Jan. 1967), pp. 12-18.
2) H. Stang and P. Southgate: Performance Evaluation of Third-generation of Computing Systems, Datamation, Vol. 15, No. 11 (Nov. 1969), pp. 181-190.
3) M. Tsujigado: Multiprogramming, swapping and program residense priority in the FACOM 230-60, Proc. 1968 SJCC, Vol. 32, pp. 223-228.
4) H. Hagiwara and H. Kitagawa: Roll in/out and usage of large capacity core memory in a time-sharing system, Memoirs of the Faculty of Engineering Kyoto Univ., Vol. 32, Part 3 (1970), pp. 348-360.
5) R. Sedgewick, R. Stone and J. W. Mcdonald: SPY-A program to monitor OS/360, Proc. 1970 FJCC, Vol. 37, pp. 119-128.
6) 北川 他：Computer-Complex による OS の Performance Monitoring，昭和 45 年度情報処理学会第 11 回大会予稿集，pp. 93-94.
7) 富岡 他：汎用コンピュータ・パフォーマンス・アナライザ，昭和 46 年度情報処理学会第 12 回大会予稿集，pp. 369-370.
8) 富士通：FACOM 230-60 MONITOR-V 解説編，システム・マニュアル，EX-041-1/EX-043-2.

(昭和 47 年 1 月 10 日受付)
(昭和 47 年 4 月 3 日再受付)