オペレーティング・システムのデザイン・クライテリア

斎藤信男（筑波大学）

1．はじめに

オペレーティング・システムは，計算機システムの発達と共に大規模化，複雑化しているが，それを設計し実現する方法論の追及が重要性をおびてきている。設計の過程に於いてはいろいろの決断が下されて進んでゆくが，目先の問題にだけ目を奪われて設計してしまうと，出来上ったシステムは大きな欠陥を有することになる。設計に際しての決断は，はっきりとした目的意識を持って高い見地から行わなければならない。

本稿では，オペレーティング・システムの設計に際してその寄り所となる規範として，抽象化（abstraction），多重化（multiplexing）およびモジュール化（modularization）をあげ，実際のオペレーティング・システムに於いて経験した問題点との関連性を論ずる。

2．抽象化と多重化

オペレーティング・システムは，与えられた計算機の資源をユーザに提供してその仕事を便利に実行させることが最大の目的である。このとき，ユーザがある瞬間に常に1人しか存在しなければ，他のユーザとの競合，相互作用等は考える必要がなく，オペレーティング・システムはハードウェアとしての資源（たとえば記憶装置，処理装置，入出力機器など）が持つ詳細な機能を抽象化または論理化してより高度な機能に変換することだけをその目的とすればよい。たとえば，入出力機器に対して直接その機器を働かす命令をユーザに使わせることはせず，駆動ルーチン（装置管理ルーチン）を設けてユーザにはより高度な機能を持った装置として使用させているのが普通である。

一方，ある瞬間にオペレーティング・システムが管理しているユーザが複数存在する場合は，同一の資源を同時に使用するという資源共有の問題が生じてくる。現代のオペレーティング・システムは，時分割システム，実時間システム，オンライン・システムなど高度なサービスを実現することを要請されているため，資源共有の問題は最も重要な点と考えられる。資源共有の実現のためには，（1）論理的に如何に正しく行うか，（2）定量的に如何に能率良く行うか，の二つの問題を解決しなければならない。

抽象化と資源共有の二つの原理は本来独立に考えられるべきものであり，オペレーティング・システムを織りなす縦糸と横糸との関係にある。資源共有を実現する一般的手法は，資源を多重化して仮想資源（又は論理資源）を作り出し，各ユーザにあてるという方法をとる。物理的な資源から多重化した仮想資源を作り出すということは，すなわち抽象化の一過程でもあるので，上述の二つの原理を一緒にして議論しがちであり，又，それで能率よく問題を解決できることもあるが，逆に二つの原理を混同しているために誤った判断をすることもある。

例1　仮想記憶

抽象化と多重化の二つの原理について具体的に考えるために，二次元仮想記憶

を考えてみよう。

物理的資源：　実記憶（線形な番地の付された記憶）

論理的資源：　二次元番地空間

線形に番地付けされた実記憶は，物理的に実装された（実際は，オペレーティング・システムの常駐部を除いた部分）記憶の大きさ以上のものをユーザ・プログラムで使用することは不可能である。一方，記憶装置を抽象化してプログラムのセグメント毎に独立の線形の番地付けが可能となる二次元番地空間を作り出す。セグメントは論理的に独立に番地付けが可能であるから，プログラム間の再配置が不必要となり，又，大きさが増減するセグメントも扱えることになり，線形の実記憶に比べてユーザは便利にプログラムを作製することが可能となる。この様に，セグメントを使用できる様にする機構を，セグメント化（segmentation）という。

一方，複数のユーザがシステム内に唯一つ存在する実記憶を共有するために，ユーザの番地空間および実記憶空間を同一の大きさのブロック（ページおよびページ・フレーム）に分割し，プログラム内である番地を参照するとき，それに必要なページのみをページ・フレームに写像してその情報を実記憶内にロードしておく。新しいページをロードする際に空いているページ・フレームがなければ，何らかのページ置き換え則を用いて古いページ・フレームの内容をスワップ・アウトし，空きを作り出す。この様な機構をページ化機構という。この様にして，複数のユーザが実記憶を共有することが可能となる。各ページ・フレームから見れば，複数のユーザに対し時間的に分割して共有を行わせていることになる。

抽象化の原理から得られたセグメントと，多重化の原理から得られたページとは，それらを能率よく実現するための一つのまとまった機能として仮想記憶とよび，一つのハードウェア機構，たとえばDAT（Dynamic Address Translator）を援用して実現している。セグメントのロードは，実際に参照する際にページ毎に行われるが，これは記憶の多重化という原理から考え出されたページ化機構を用いて，記憶の抽象化という原理から考え出されたセグメントの実現を行っていることになる。一方，複数のユーザが同一の情報を共有するとき，一般にセグメントを介して行った方が柔軟性に富み便利である。記憶の抽象化という原理から考え出されたセグメント化機構を用いて，記憶の共有を実現していることになる。

以上に述べた様に，仮想記憶においては抽象化という原理と共有化，多重化という原理が相互に組み合わされて，便利な論理資源を実現している。

例2　TSSにおけるラインプリンタ

TSSシステムにおいて，図1の様に端末からオンライン的にラインプリンタを使用する場面を考える。

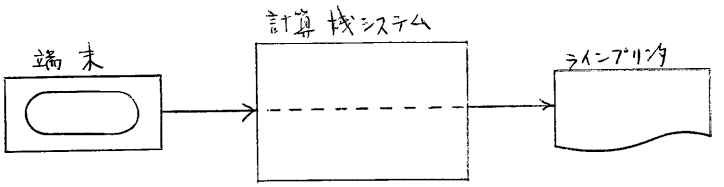

図1　TSS端末からのラインプリンタの使用

この様なシステムを設計する際に，その仕様を決めるため

に種々の判断を下すが，どの様な原理に従って判断を下すかにより，種々の場合がある。

① ラインプリンタは，本来バッチ処理をしているプログラムが使用するものであり，TSSの端末からは一切使用させない様にシステムを設計する。

② ラインプリンタは，1人のユーザが使用し始めたらその仕事が終了するまでレコードが連続しているものである。従って，既に作られているファイルの内容を出力させる様な要求だけを端末から受け取り，その待ち行列に従ってラインプリンタを駆動させる様にシステムを設計する。

③ 計算機システムに蓄積されている機器は，ユーザが自由にその機能を使用できるべきである。従って，端末の画面に表示されるメッセージ（TSSシステムのコマンド，プログラムの編集，プログラム実行時の入出力データなど）でも，その任意の部分をラインプリンタに出力することが随時可能となる様にシステムを設計する。

以上，3つの場合についてシステム設計に於ける決断と，抽象化，多重化の原理との関係を論じてみよう。

①の場合：

ラインプリンタという物理的資源の多重化という点のみを考えると，時間的に順次使用するのがラインプリンタの物理的性質であることから，バッチ処理に於ける多重化は簡単に実現できるが，オンラインとして要求の到着の仕方がランダムである様な使用を許す多重化の実現は複雑になると予測し，オンライン的な使用を禁止したことになる。この場合，多重化の実現方法が対象となる物理的資源の性質に大きく左右されたことになる。

②の場合：

端末のユーザから見れば，ラインプリンタという物理的資源が計算機システムに装填されているので，何らかの形で使用してみたいと思うのが普通である。その際，ラインプリンタの持つ機能を少々制限しても良いという妥協が成り立つ。すなわち，抽象化する際に立脚すべき物理的な性質を本来備えている性質より制限されたものとして考えておく。こうすると，多重化の実現も比較的能率よく行えることがある。既に順ファイルとしてファイル装置内に格納されている内容を，端末からの要求の待ち行列に従って出力するわけであるが，ラインプリンタの起動は，自動的に行うかまたはコンソール・オペレータからの要求によって行わせることができる。プログラムの実行によってもたらすデータの出力は，一旦ファイルに書き込むことをユーザが行い，それを端末から指定して出力するという間接的な方法をとらなければならない。この場合，多重化の実現を容易にするために，抽象化の原理に制限を加え，妥協した形でシステム設計の決断を下したことになる。

③の場合：

1人のユーザが計算機システムを専用的に使用しているとすれば，システム内に装填してある物理的な資源はその物理的機能を全て使用できることになる。そこで，抽象化に際して，その立脚すべき物理的性質として，ラインプリンタが持つ物理的機能全てをとることができる。これが，③の場合に相当する。従って，端末の画面に表示されるメッセージも，必要であるならば随時ラインプリンタに出力させることが可能でなければならない。勿論，この場合出力結果をファイルに書き込むなど間接的な操作をユーザが考慮することは必要としないことが前提となっている。抽象化の原理だけを追及してゆけば，この様な論理的資源に到達するであろう。一方，このような抽象化を行った論理的資源を多重化することは，かなり複雑な問題となるであろう。たとえば，端末毎にラインプリンタに相当する順ファイルを用意し，一旦そこに仮想的な出力を自動的にしておき，適当なきっかけを作ってその内容を実際のプリンタに出力させる様な方法をとればよい。この場合，抽象化の原理と多重化の原理とを全く独立に追及してシステム設計に於ける決断を下したことになり，実現の能率の問題を除けば，ユーザにとっては使い易く便利なシステムとなる。

3．柔軟性とモジュール化

ソフトウェア・システムは，一旦作製された後で，エラーの修正，仕様の変更，環境の変更，ユーザの要求などの要因により，しばしば修正を受けることになる。この場合，その修正が柔軟に行える様に設計・実現をしておかなければならない。オペレーティング・システムの様に巨大で複雑なソフトウェア・システムに於いては，その配慮を充分にしておくことが特に重要である。

柔軟性を持たせる方法は，徹底したモジュール化を行うことである。修正が一部のモジュールだけで済めば，非常に能率良く実施することができる。この場合，モジュール化したというだけで，修正が全体のモジュールに及ばないという保証はない。どの様な原理に基づいてモジュール化を行うかによって，それは左右される。

モジュールに分割するためには，いろいろの規範が考えられる。たとえば，

(1) 機能
(2) 抽象化レベル
(3) 実現方法

などがその規範となるであろう。これらの規範のうち，どれをとっても同じモジュール化ができればよいが，Habermann [1] が指摘している様に，モジュールとレベルとは必ずしも一致しない。これらの点を考慮に入れて修正しやすい（柔軟性のある）オペレーティング・システムを設計することは，今後ますます重要となるであろう。

例3　機器の切り替え

あるミニコンピュータ・システムに於いて，図2に示す様に，タイプライタとCRTディスプレイ（タイプライタ互換性のあるもの）とを購入した。ミニコンのオペレーティング・システムは，コンソールだけから入出力を行うバッチ処理システムである。一方，そこには大型機のTSSの通信回線が来ており，その端末装置としてタイプライタまたはCRTのどちらでも使用できる。そこで，次の

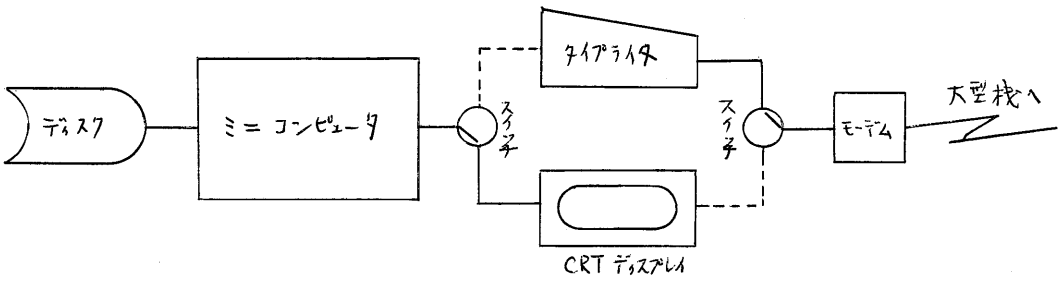

図 2　コンソール の切り換え

様な使用方法を考えた。

（1）ミニコンでプログラムを作るとき，CRT の方が使い易いので，CRT をミニコンのコンソールとして，タイプライタを大型機の端末として使用する。

（2）ミニコンにはハードコピーをとる装置が用意されていない。ハードコピーが必要なときは，CRT とタイプライタとを交換し，タイプライタをミニコンのコンソールとして，CRT を大型機の端末として使用する。

図2におすような切り換えスイッチをハードウェアとして用意することは容易にできるであろう。それで万事解決し，二つの端末機器を効率よく使いこなすことができるかと言うと，そうではない。ミニコンのオペレーティング・システムの都合上，ミニコン内の I/O インターフェースのカードごと交換しないとその様な使用方法が不可能であると知らされ，何故かと不思議に思った。オペレーティング・システムの内部構造まで分析してみなかったので，そうしなければならない理由は正確にわからないが，コンソールの入出力ルーチンが，機器の物理的な番地や性質まで取り入れて設計されていることが予想される。本来ならば，コンソールの入出力ルーチンを二つのモジュールに分割し，一つは論理的なコンソールへの入出力ルーチン，もう一つは具体的なコンソールの機器の物理的な性質を処理するルーチンとして設計し，実現しておけばよい。具体的な機器の指定を行えば，それに応じて物理的性質を扱うルーチンを適当に選択すればよいはずである。

例 4　オペレーティング・システムの常駐部

ある計算機システムのオペレーティング・システム（仮想記憶のサポート有り）の常駐部が，実記憶容量の 80 ～ 90 % を占めてしまい，そのために残りの実記憶を用いて仮想記憶を実現するので，非常に能率の悪いシステムになってしまうことがあった。仮想記憶のサポートをするために，オペレーティング・システムの規模が増大し，その結果常駐部が増大することはよくあることである。また，仮想記憶を用いるので，論理的には，どんなに実記憶が小さくても，ユーザ・プログラムの処理をすることはできる。しかし，現実には，その様なシステムは使いものにはならないであろう。仮想記憶の機構を備えていれば，記憶の大きさなどは一切気にしなくても良いはずであったのが，皮肉にも，記憶容量のために却っ

てシステムが使いにくくなってしまうという事態が生じるわけである。

オペレーティング・システムの常駐部の大きさに関しては，仮想記憶をサポートしていない場合でも，いつでも気になることである。一般的に使用される機能を全てサポートする様に常駐部が構成されるため，それが増大する傾向にある。前述の例に於いては，単一のユーザが使用する場合と，多数のユーザが使用する場合とで，常駐部の大きさの差は殆んどないであろう。従って，単一ユーザの場合，使用しない部分が常駐部に多数組み込まれていることになる。オペレーティング・システムの設計が徹底的にモジュール化されていれば，実際に使用する場面に応じて必要かつ十分な機能だけを常駐部に組み込むことができるはずであり，無駄な記憶の使用もなくなってくる。その様な意味では，現状のオペレーティング・システムは，あまりにも柔軟性が少いと言えるであろう。

4．おわりに

オペレーティング・システムの設計に際しては，ある規範にのっとった決定を下さないと，非常に使いにくいシステムが出来上ってしまう。その規範として，抽象化，多重化，柔軟性をあげ，現実のシステムに於ける問題点との関連を論じた。これらの規範は相互に独立の概念であり，それらを混同せずに徹底的に追及すれば，良いシステムの設計ができるであろう。

文献

[1] Habermann, A. N., Flon, L. and Cooprider, L. Modularization and Hierarchy in a Family of Operating System, CACM, Vol. 19, No. 5, pp. 226～272, 1976