OS の 展 望*

竹 下 亨**

1. はじめに

オペレーティング・システムは，マスタ・コントロール・プログラム，システム・モニタ，エグゼカチブ・プログラムなどともよばれているが，その初歩的な概念は，きわめて原始的なモニタ・システムであるSummer Session Computer and Utility System が MIT で使われた 1953 年にさかのぼることができるといわれている．その後 1950 年代の後半には，いくつかのごく簡単な OS がユーザーにより作られた．汎用性の高い大きなものとしては，1960 年に世に出た SHARE Operating System が有名である．そして第3世代の計算機に使われている最初の本格的な OS が姿を現わしたのは 1966 年3月であった．

第2世代の計算機に使われた最大の OS は 414,235 個の命令より構成されていたが，現在存在する大きな OS はその 10 倍近いであろうと推定される．この数値が示すように，いまや汎用の OS は巨大ともいえる規模になっており，専門家でもその中味を知りつくすことはきわめて困難になっている．そのごく一部のみを知り OS の全体を評価すること，たとえていうならば，樹を見て森を見ずのことも少なくない．そこで，ひろくその全体を見わたして OS を論じるようにということで，「OS の展望」というテーマが私に与えられたと了解している．今後どのような方向に発展するかを大いに知りたいところだが，現在利用可能な OS がどんなものかを再確認し，これを使いこなし，また問題点があれば，それを改善することがまず重要であると思う．

以下に，OS を必要とする背景，OS のねらい，生い立ちを振り返り，OS の基本機能，特徴，最近に開発された機能を検討し，進歩・発展に現われた傾向をさぐり，そのねらいがいかに達成されているかを評価し，将来の見とおしを考察しよう．

昭和 45 年 12 月 11 日開催の第 11 回大会における招待講演

* A View of Operating Systems——Their Past, Present and Future by Toru Takeshita (IBM Japan Ltd.)

** 日本アイ・ビー・エム

2. 背景と意義

計算機の利用が進み，手作業を単に機械化するにとどまらず，高度かつ複雑なアプリケーションが多く見られるようになり，データ処理に大きな投資がなされるようになると，必然的にハードウェア，情報，人的資源の有効な使用——とくに，CPU を遊ばせないようにすること——が要求されてきた．そして，人間はより知的かつ創造的な仕事ができるようにしたいというわけである．

そこで，言語翻訳，資源管理，情報検索，作業計画と監督，機械装置の操作，管理などを可能な限り自動化し，それにより DP インストレーションの生産性向上を高めることが考えられ，OS はそのために設計されているといえる．その主目的は，第一にスループットやターンアラウンド・タイムで代表される性能 (performance) と利用可能度 (availability) を上げ，第2にプログラミングの補助や各種のサービス・ルーチンやその他の共通性のあるルーチンによるプログラミングやスケジューリングの単純化・迅速化や，人間とシステムとの相互反応の向上などシステムを使いやすくするための便宜 (facility) を多く提供することである．

3. 発展の経過

現代の OS は，ハードウェアの世代とほぼ平行して，三つの段階を経て発展してきたとみることができるが，それは人間と計算機とのギャップを埋めるための発展である．

第1段階においては，OS に使われる種々の**構成要素が開発**されたといえる．すなわち，計算機とプログラマとの間の言語の相違を解決するため，翻訳プログラムを初め，IOCS，その他のプログラミング補助が提供され，共通に使われるサブルーチンや汎用化プログラムが作られ，プログラム・ライブラリが設けられるなど，プログラムやデータの共用が進められ，これと同時にアプリケーションも急速に進歩した．

つぎの段階では，計算機がトランジスタ化されて，

高速・高性能となり，大型機種ではオペレータが計算機についていけなくなり，両者の間の速度・信頼性・反応時間などのギャップを少なくするために，第1段階で作られた**構成要素を統合化**し，オペレータの介入を少なくして，独立なジョブを連続的に処理する OS が作られることになった．その基本目標は，多数のジョブのノン・ストップ処理であった．こうした OS を実現するのに，バッチ・ジョブ処理 (batch job processing) すなわち，積重ねジョブ処理 (stacked job processing) を可能にするため，共通のジョブ入力装置，ユーティリティ入出力装置，ジョブ出力装置が定められ，かつ，制御プログラムや使用頻度の高いプログラムはシステム・ライブラリに入れ，他方プログラムとオペレータに対するコミュニケーションのための新しい制御言語が作られた．かくして，自動的にジョブのステップからステップへの移行をさせるシステムができあがった．この OS の核 (nucleus) となる部分は常に主記憶装置に常駐し，制御プログラムの中の一時的に必要なルーチン——制御ステートメントの読込み，解読，実行のルーチンなど——は，それが必要なつど，読み込まれることになった．こうした OS は，当初はどちらかというと，科学技術計算のジョブを対象として設計され，読み込まれたジョブを順次 (sequential) に処理するテープ・システムであった．

丁度このころ，こうしたバッチ処理システムとは全く独立に，リアル・タイム・システムなるものが誕生していた．その初期のものとしては，軍事上の目的より作られた SAGE (Semi-Automatic Ground Environment) や人工衛星のためのマーキュリ・システムや SAGE の商業版の SABRE (Semi-Automatic Business Environment Research)——アメリカン航空の座席予約システム——が有名であり，日本においては汎用の計算機を使用した最初の大規模なシステムでは，東京オリンピックで使われたものがあり，これらのために大きな制御プログラムが作られた．その後このオンライン・システムは，銀行の預金システムを初め多くの分野に普及している．オンラインのシステムでは，データの発生源と送り先が CPU と直接つながっている．このシステムがもつ大きな技術的問題は，入力データがランダムに発生するので作業負荷が動的に変化することであり，これを並行的な処理 (concurrent operation) で解決している．すなわち各取引き (transaction) を別個の異なる作業単位として取り扱い，しかも複数の取引きを同時的に処理している．間もなくこのようなやり方がバッチ処理の OS に取り入れられないだろうかということに気がついたのである．

OS の**第3の段階**の発展は，**手法の結合**ということが大きな特徴であるといえると思う．すなわち，第2段階で開発されたバッチ処理システムとリアル・タイム・システムの手法を改善し，これらを結合・統合して，高度化したいくつかの処理方式をカバーする汎用の OS を作り出した．初期のバッチの OS に存在したプログラム設計の方法やジョブの管理やデータの管理は拡大・強化され，他方，タスクの並行処理なる概念や手法はリアル・タイムのシステムより導入された．こうして，OS の適用範囲が拡大し，全体の生産性を著しく向上させ，種々の高度のアプリケーションの開発とオペレーションをしやすくするシステムとなった．

4. 設計目標

汎用の OS の設計目標を ここで ごく簡単に整理しておこう．

その第1は何といってもターンアラウンド・タイムの短縮，スループットの増大，プログラマの能率向上，オペレーションの効率化などを可能にして，計算機システムの全体的能力を上げることである．

第2は，多種のユーザーのニーズを満たす汎用性をもたすことで，このために積木方式 (modularity) の設計がなされ，ほとんどの OS はユーザーの機械構成やその他処理上の必要に応じたシステム生成を導入時に行ない，かつ実行時に指定できる機能や方式のある程度の選択項目 (option) がある．

第3は，拡張性 (expandability) や 開放性 (open-ended) であり，機能や能力の増大を容易にし，アプリケーションの拡大に応じ，かつ技術の進歩を組み入れてゆきやすいように設計されていることである．そのほか，拡張や成長をしやすくするには，互換性，装置より独立，多重タスク管理，データ形式やラベルの標準化などが重要な要素である．

こうした高尚な設計思想は，程度の差こそあれ，ほとんどの第3世代の計算機の本格的な汎用の OS に取り入れられており，その存在価値を高め，生命を長くさせるのに役立っているといえよう．

5. 構成とおもな機能

現在のほとんどの OS はマルチプログラミングを行なうことが可能なように設計されている．そして，並

行処理できるジョブの数が固定されているものとこれが動的に可変なものがある．その制御プログラムの機能は，メーカーや機種によって，分類や名称が異なるが，大別すると，1）**スケジューラ**によるジョブの読み込み開始・終了，2）**スーパバイザ**による計算機の中での仕事（task）の進行の監督，および 3）**データ・セット（ファイル）の制御**と**データのアクセス**である．

スケジューラの仕事はジョブ管理ともよばれ，その主要な任務はノン・ストップのジョブ処理である．その基本的なものは，先にはいったジョブが終わってからつぎを始める順次ジョブ処理であり，現在では珍らしくなくなっているのはマルチジョブ処理である．それには，ジョブの処理が進行中につぎのジョブを読み込み，前のジョブの出力を書き出すといったシステム・スプール（system spool）もある．マルチジョブのシステムでは，処理の開始ははいってきた順ではなく，与えられた優先度に従うスケジューラがあり，かつどのジョブを主記憶装置のどの区画に入れるかを，そのつどオペレータが判断しなくてもすむように，ジョブ・クラスを与えることを許すものもある．なお，最近では，遠隔端末よりジョブを計算機に送り込む遠隔ジョブ入力（remote job entry）を行なう OS がいくつか存在する．

本格的な OS の特徴の一つは，一時に複数のタスクの実行をスケジュールし，その実行を監督できることであり，それはシステム資源の効果的な管理によって行なわれる．プログラムやデータをできるだけ共用するよう種々の配慮がされているが，たとえば，一つのルーチンが並行的に進行するプログラムにより同時的に使えるようにするには，再入可能（re-entrant）な形に組まれる．

スーパバイザはシステム資源をタスクの要求に応じて動的に割り当て，複数のタスクの実行を管理することにより，システムの効率を増大させ，ターンアラウンド・タイムを短縮し，システムの融通性を高め，その拡大を容易にしている．なお，主記憶装置の区域については，システムによっては，単なる上重ね（overlay）だけではなく，掃出し/掃込み（rollout/rollin）や記憶交換（memory swapping），あるいはページめくり（paging）を行なって，複数のユーザーの同時的使用や大きなプログラムを分割してコーディングしなくてもその収容を許すなど，主記憶装置のスペースの有効利用を可能にしている．また，制御プログラムなどが他のプログラムよりこわされないような記憶保護（memory protection）もなされている．

増大するデータとその中央集中化の問題を解決すべく，データの編成，その入出力方式，ライブラリ参照システムなどに統一的な管理の方式が導入された．これにより，同一の OS の下に働くプログラムは同じ種類の方式でデータをアクセスすることになり，ある意味で標準化された．他方，データの形や装置などを実行的に定義することを可能にして，いわゆるプログラムの「装置と独立（device independence）」を実現した．さらに，その上に，後で触れる“データの形と独立”なデータ管理システムも出現しているのである．また，合言語（password）などの手法を使用したデータの保護も行なわれている．

オンライン処理に対して，データ通信の入出力制御のルーチンが組み込まれ，それも，回線の制御を主とする基本的なものから，メッセージの待合せやその種類の解読，必要なルーチンへの引渡しを行なう汎用のメッセージ制御プログラムが使われている．

また，陰極線管（CRT）が入出力装置として利用され始め，映像データ処理が行なわれるようになり，これを支える入出力関係の各種ルーチンが組み込まれている OS も少なくない．

先に一つの CPU で複数のプログラムを並行的に処理するマルチプログラミングについて言及したが，複数の CPU で一つのシステム——主従システム，ファイル共用システム，主記憶装置共用システム——を構成する OS も存在している．これらのねらいの中にはデータの共用，資源の有効利用，生産能力の増大があるが，多重 CPU で可変数のタスクのマルチプログラミングを行なうシステムはリアル・タイム・システムに要求される利用可能度をあげるのに大いに役立つものである．このような理由から最近のいくつかの大型機は，マルチプロセサ・システムとして設計されている．

こうした複雑な高度のソフトウェアの開発に際しては，GPSS や SIMSCRIPT や CSS やそれ以上の高度化したシミュレータが使われて，ハードウェアとソフトウェアの機能的要求，相互反応，構成要素の効果度，システム構成と全体の性能，最適化などが調べられ，設計上に役立てられている．

6. タイム・シェアリング・システム

計算機の効率的利用方法として，タイム・シェアリ

ングが注目され，アメリカでは 1960 年代の後半に急速に発展したが，わが国でも今後普及すると考えられている．そこでこれについてとくに取り上げて，少し述べる必要があると思う．

この思想は 1950 年代の中ごろに生まれており，最初のシステムは 1961 年 11 月に MIT の計算センタで公開された CTSS であるとされている．タイム・シェアリング・システムは不特定多数のユーザーに端末をとおして，対話方式で計算機の利用を可能ならしめいることが大きな特徴である．

時間の刻み (time slice) の割り当ての仕方は，システムの性能を左右するが，簡単なラウンド・ロビン方式かこれを修正したものや，テーブル・ドリブン・スケジューラなどが使われている．複数のユーザーのタスクを並行的に処理するための主記憶装置のスケジューリングには，単なる記憶交換があり，またページめくり手法を使った仮想記憶 (virtual memory) で動的配置替え (dynamic relocation) を行なうものもある．このほか TSS に重要なソフトウェア機能としてはファイル管理，指令言語，対話方式の言語がある．

最近の大型機の OS には通常のバッチ・ジョブ処理，遠隔ジョブ入力，および いま述べたタイム・シェアリングの機能をもつのが普通となっており，これを 3D (次元) システムと呼んでいるメーカーもある．

タイム・シェアリングは計算機の利用を高める様々な方式を生み出すことになったが，その一つとして注目されるのは，計算機のシステム資源をさらに高度のレベルで管理し，各 (遠隔) ユーザーがあたかも完全な専用の計算機として，その思いどおりに使えるようにしたタイム・シェアリングのマルチアクセス・システムである．この概念は仮想機機 (virtual machine) と呼ばれ，いくつかの異なる OS を並行的に走らせることができるので，各ユーザーはそれぞれ別個の OS を選んで同時的に使うことが可能である．

7. 最近のトピックス若干

最近の OS のことについて概観してきたが，まだ触れていない話題のうち，おもなもの若干を述べることにしよう．

OS の機能が増大し，性能が向上し，オンライン・ジョブを含めたマルチプログラミングが行なわれているシステムでは，システムの故障や一時的な中断は，企業の経営に重大な支障をもたらすことになる．そこで，フェール・セーフ (fail safe) やフェール・ソフト (fail soft) にするなど，信頼性や利用可能度をあげるのに大きな配慮が払われるようになっている．すなわち，自動診断ルーチン，オンライン・テスト・プログラム，自動切替ルーチン，回復管理ルーチン，高度のチェックポイント/リスタートなどが作られている．

増加する入出力装置の種類や新しいハードウェアの機能およびアプリケーションの様々な要求に応じて，OS の機能は拡大を続けているが，他方，すでに開発されたものを使用したうえでの，効率化・最適化や使いやすさのための改善が継続されている．卑近な例は，使用頻度が高いルーチンは主記憶装置に常駐させるように変えるとか，処理時間が短くコンパイルの回数の多いジョブにはリンク・エディタの代わりに効率のよいローダを使うとか，コンパイルする前に文法的誤りを検出する構文チェッカを組み入れるなどである．

オペレータの効率をあげ，その負荷を軽減することもされている．その代表的な例としては，映像表示装置を使用した操作卓であり，また，多数の入出力装置をもつシステムでの複数の操作卓にメッセージを振り分けて出す仕組みである．

マルチプログラミングの効率はジョブの組合わせ方と優先度の与え方によって大いに変わってくる．これをいちいちプログラマやオペレータが決めるのは面倒であり，このことを何とか自動化できないかと考えるのは当然である．ジョブ・クラスを各ジョブに与えておいて，それによって大きさや優先度が異なる区画にはいるようにすることは，この問題の一部の解決とはなる．しかし，それよりも一段と進んだ方法として，ジョブが中にはいってから，CPU と入出力の使用の度合を見ては自動的に優先度を変え，CPU バウンドのジョブと入出力が多いジョブとが並行するようにする自己発見的 (hueristic) な手法が開発され，利用されていることは大変興味深い．これが発展すれば，ジョブをあらかじめ数多く入れておけば，システムが自動的にそれらの優先度を適当に変え，システム資源がもっとも有効に使われるように，ジョブの実行の組み合わせを動的に変えることになる．

ジョブごとに使用した CPU 時間や入出力の量を記録する会計ルーチン (accounting routine) は第 2 世代の計算機の OS より使われているが，これよりもずっと細かい情報——ジョブやジョブ・ステップの開始，終了時刻，プログラムで使われた領域や区画の大きさ，入出力指令の実行数，データの量等々——を自動的に収集・記録するシステム管理機能 (System Manage-

ment Facility) なるものが開発されて，計算機内の資源の利用の度合を知り，それによりシステムの構成やジョブの組合せなどを修正して，一層効率よくシステムを働かせることを可能にしている．これが大いに進歩すると，計算機を導入してある期間後に，どの装置は余り使わないから返却すべきだ．どの装置をふやすと CPU の利用度やシステム効率がどれだけ上るというようなメッセージが出るようになるかも知れない．

なお，OS の制御プログラムそのもののことではないが，そのとくにデータの管理関係の機能の拡張・拡大とみなすこともできる汎用のファイル処理システムや情報（管理）システムなるものが既存の OS をベースとして開発され，実用化されており，本格的な OS が総合的データ・ベースやデータ・バンクのシステムを作成するのに基盤として役立っている．

また別の形の OS の機能の拡大の例に，つぎのような問題向き言語解析システムがある．各特定のアプリケーション向きの言語に対して，別個にコンパイラを開発しなくてもすむように，言語の辞引と処理モジュールを収容していて，入力プログラムの言語を解析して，その実行に必要な処理モジュールに結びつけることを行なうようなシステムであり，すでに二つの種類のもの——PLAN (Program Language Analyzer) と ICES (Integrated Civil Engineering System)——が知られている．

以上述べたことを含めて，OS に関して様々な動きがあるが，各メーカーが力を入れているのは，やはりデータ通信やタイム・シェアリング関係の機能であろう．前者では，メッセージの待合せの進んだ方式を有し，回復や保守が容易で，診断補助機能を備え，一つの端末からの複数の異種の仕事の入出力を許すような高度，かつ総合化された遠隔通信入出力制御システムが実現している．それに加えて，工場の自動制御や風洞実験の監視などのための実時間能力も付加されている．

タイム・シェアリングのシステムでは，これまでのノウハウや実用経験を結集して，汎用機に使われる汎用の OS に効率的な TSS 機能が組み込まれることになった．従来は，主として科学技術計算用に設計された専用のシステム (dedicated system) があり，対話型の FORTRAN, BASIC などが使われており，同一計算機でバッチのジョブも並行的に処理する場合の言語やファイルの互換性は重視されなかった．しかるに，最近の大きな OS では，タイム・シェアリングのジョブと背景処理とが並行処理され，かつ両者に言語やデータ・セットに互換性があり，バッチのジョブとタイム・シェアリングのジョブが同一データ・ファイルを何らの支障なく共用することや，タイム・シェアリング方式で端末より（バッチの）プログラムを開発することなどが行なわれている．そして，端末のオペレータと計算機との間のやりとりをしやすくするために，強力かつ便利な指令言語 (command language) が開発されているし，それに，端末のオペレータがなすべきことをメッセージで次々と指示して，言語翻訳ルーチンと端末のユーザーとの相互反応を促進するもの (prompter) も考えられた．かくして，専用の TSS より始まり，今や文字どおり汎用性と互換性のある TSS が実現している．それは，バッチ処理，単なるオンライン処理，遠隔ジョブ入力処理，タイム・シェアリング処理が一つのシステムで同時的・並行的に進行する高次元の総合システムとなっている．

大きなシステムで開発され，実用化されたものが，小さなシステムにおいて実現することが少なくない．タイム・シェアリングのシステムでもしかりである．専用のシステムでなくとも，端末との効率的な相互反応機能を主記憶装置のスペースをさほど要せずに提供することにより，小規模な汎用の計算機で汎用の OS の下にタイム・シェアリングを行なうことが可能になってきている．かくして，小規模なユーザーにおいても，小さな計算機で通常の事務計算を行なわせ，それと並行して，技術者らがそのシステムをタイム・シェアリングで使用することも次第に行なわれることになろう．

技術の進歩によりハードウェアの世代が数年ごとに変わるが，旧世代のプログラムはそのまま実行できないことがある．またそうでなくとも，他の機種に切り替えたときに同様なことが起きる．多くのプログラムの開発には人件費や機械使用の費用などにより，大変なコストがかかる．そこで，ユーザーがソフトウェアの開発に投資したものを保護すること，つまり，プログラムを書き替えずに使えるようにすることが望まれる．このため，第1世代より第2世代に変わったときには，シミュレータが開発され，第3世代になると大部分の機能をハードウェアにさせるエミュレータが開発されたことは周知のとおりである．このエミュレータを OS の下に働かすようにして，旧世代のプログラムを混入して連続処理することが行なわれているし，同一ファミリーの計算機で複数の OS があるときに，

小さいほうの OS をエミュレートする機能を導入して，小さい OS 用に作られたプログラムを大きい OS 用に書かれたものとマルチプログラミングすることも可能にしている．これは，小さい OS から大きい OS に切り替えるユーザーに便宜を与えるのである．見かけ上，一つの計算機が複数種類の計算機として動くことになるわけである．

こうして見ると，現在や今後の汎用の OS は数年前に予想もしなかったいろいろな機能を含むようになり，その全体的能力が大幅にあがっていることが認められるのであるが，このような大規模，かつ複雑なシステムを開発するには，ぼう大な人員，機械時間，日数，コストが必要とされる．それがゆえに，より厳密な計画と進歩した進行・工程管理が実施され，厳格な品質管理が行なわれ，保守の体制も大幅に整備されている．なお，ソフトウェアの開発を効率化するため，高水準のシステム言語を使用されているし，モジュール間の結び付け (interface)，互換性，修正・拡張・保守のしやすさなどを考えたうえの標準化も随分進んでいる．

8. 最近の利用状況

さて，この辺で話題を変えて，OS がユーザーによりどのように使われ，どのような評価を受けているか検討してみよう．

第3世代の OS が発表されたころは，OS は“万能であり”“何でもできる”“きわめて容易に使える”といわれ，とくにマルチプログラミングがいとも簡単にできるように PR され，それを信じたユーザーも少なくなかったと思われる．しかし，いざ OS を始めてインストールして見ると様々な問題があった．まず，前世代とはあまりにも進んだ高度の機能をすぐに理解し，思うように使いこなすことは容易ではなかった．たとえば，オペレーションにしても，ボタンを一度押せばあとはシステムが自動的にすべてやってくれると考えた人もあったが，実はオペレータは前世代よりも高度かつ長時間の勉強を必要としたのであった．数学にたとえれば，小中学校の算数から高等数学に一足飛びしたような感じであった．マルチプログラミングにしても理論的には可能であり，OS はその機能を持っていても，実際にやらせてみると，主記憶装置のスペースや入出力装置が十分でないことがわかったり，予想した性能が出ないなどの問題が経験された．

しかし，いまジョブを次々とノン・ストップで処理させること自体は当然のこととなり，OS の高度の機能・能力を十分理解し，これを使いこなしているユーザーが多くなっている．SPOOL 的な使い方が多いとはいえ，かなり小規模なシステムでもマルチプログラミングがすでに定着しているといえよう．現在の OS の最大目標には，積木方式，共通のインターフェース，統一的データ管理，ジョブ制御言語等々をとおして，トータル・システムとしてユーザーのアプリケーションを統合化しやすいようにすることであり，また，マルチプログラミングによって計算機内の資源をできる限り遊ばせずに効果的に使うようにして，計算機の使用効率を最大にすることであった．

これらは，いまや理論 (theory) として考えられた段階から真に実用 (practice) 化されており，初期に見られた実験 (experiment) や試み (trial) に近いことから，ユーザーにおける実際の生産 (actual production) に大きく役立ち，ユーザーの実質的な利益となり，その満足を得ている段階に到達したと多数の人達が評価している．いまや 1964 年に第3世代の計算機が発表されたとき，そのソフトウェアとして考えられていた概念や期待がまさに実現していると考えてもよいと思う．

現在のユーザーの多くは，OS の高尚な概念をマスターし，実際の仕事に応用した経験を積み，新しく加えられる高度の機能を理解し，これをさらに高度化もしくは効率化のために活用していこうという積極的意欲を示している．そして OS をベースとした速い応答速度のシステム (fast response system) やデータ・ベース中心とするシステムなどにより，総合的な MIS の開発に役立てているケースが多く知らされている．

このような情勢にあって，ここ 2，3 年わが国でも顕著になってきたのは，ユーザーのシステム選定に OS を中心とするソフトウェアを重視していることである．このため，OS の機能・性能が詳しく比較・検討されるのみでなく，マルチプログラミングを行なったときの能力を重点的にテストするいわゆるベンチマーク・テストが行なわれることが少なくない．

しかも，大型になればなるほど，導入されたシステムで実際のジョブを流したときの性能の分析・評価が，多くの角度から行なわれているようである．この結果によって，機械や制御プログラムの一部を変更したり，ジョブの組合せや入れ方を変えること――整調・調律 (tuning)――を行なう．そして，こうした性能分析には，特別なハードウェアの装置をつけてモニタした

り，前述したSMFをソフトウェア・モニタとして活用している．

また，ごく最近の報道によれば，本邦においても，システム・タスクの負荷量，バッチ処理のジョブへの影響量，ユーザーのジョブに使われているCPU，主記憶装置，ファイルの使用度合をシミュレーションによって求め，仕事の量と質に応じて，最適の機械構成を，実際に機械を入れなくても考えることができるシミュレータがユーザーにより開発されている．これによりCPUに対し，どのような端末機をどのくらいおいてサービス通信網を組んだらよいかを知ることができよう．

9. おわりに（当分の見とおし）

以上でOSの過去と現在を大ざっぱに概観した．そんなことはどうでもよい．これから何が出てくるのか，何が将来開発されるのか聞きたいという向きも少なくないかも知れないが，大変大きな複雑なものになっているOSの今後の見とおしを考えたり，理解するのにすでに開発された高度なものがわかっていることが必要であり，それなしには新しいものの評価はむずかしい．

超大型機に見られるパラレル・プロセサやアレイ・プロセサなどの新しい種類のハードウェアを使用したシステムやプロセス制御，交通制御など特殊目的のシステムでは，それなりにOSはかなり異なったものになる．しかし，一般に使われる汎用のOSについては，ごく近い将来に全く革命的なものが出現すると考えている人は少ないようである．その理由は，現在のOSが，きわめて多種類のアプリケーションに適合し，広範囲なハードウェアの構成を支えるように意図された積木方式のシステムであり，開放方式の設計がなされていて，必要となった新しい機能が容易に組み込まれるようになっているからである．すなわち，基本的概念は長い生命を目標としており，これがかなりうまく達成されているように考えられるので，全く概念の異なるハードウェアが出現しない限り，現在のOSと画期的に異なるものは近い将来現われないと見る人が多い．全く新しいシステムを開発するとすれば，メーカー側もばく大な開発費を投ずることになるし，リスクも大きい．しかもユーザー側にして見れば，これまでに作成したプログラムを新しいシステム切り替えなければならないとすれば，これまでのプログラミングへの投資がムダになり，変換に大きなコストをかけることになる．第3世代の計算機が納入され始めたころの切換えのための多くの問題や大変な作業量を経験されたユーザーは大幅な変革に好まないのではなかろうか．大多数のユーザーはコストが安くて，生産性や互換性が高く，使いやすいシステムを要求している，とすると，大多数のユーザーに受け入れられる汎用で，しかも各ユーザー固有の要求に応じて多少の構成上や機能の選択が許されるOSが望ましいことになる．

OSがハードウェアの延長として不可分であり，システムの選択に際して，それにより重点がおかれるようになると，いままでより進んだ需要の調査・検討・予測が行なわれ，これまでの経験の蓄積のうえに開発技術も一段と前進することが期待できる．

これまでのOSに取り入れられた概念や手法は，取捨選択，改良されてゆくし，新しい概念や機能——新しい装置やハードウェア機能に対応するものに当然として——の導入が行なわれるであろう．

これからのOSの新しい機能の中で重視されるのはデータ通信関係であり，取引中心（transaction oriented）の処理方式が指向され，人間との相互反応をより向上させ，記憶交換や動的配置替えなどのやり方も進みタイム・シェアリングを最適化するものや，映像表示装置のより多くの活用が見られるのであろう，また，小型機を知的端末（intelligent terminal）として使う通信網を制御して，負荷のバランスをうまく配分するOSの機能も伸びるであろう．

もう一つは，データ管理関係の機能の発展であろう．すでにリスト構造のファイル編成の高度のシステムが使われているが，これまでのデータ編成や入出力アクセス方式の実積のうえに立って，それらを効率化，もしくは高度化が考えられるし，データ処理の総合化を進めるためのデータ・バンクやデータ・ベースを支えるソフトウェアの強化は必然的であろう．

当然ながら，マルチプログラミングをより効果的にするための一層の改良と発展が予想される．また，最近発売された大型機はマルチプロセサとして設計されていることから，この方式をカバーするOSの機能はさらに前進すると思われる．

こうした機能の拡大・増強と同時に機能の向上は相変わらず続けられるであろう．記憶装置が大きくなり，アクセス・タイムが速くなり，1ビットあたりの単価が下がれば，OSのオーバヘッドは相対的に減ってきて，単価あたりの性能（price performance）や全体的コストの低下にもなる．

しかも，RAS，すなわち信頼性 (reliability)，利用可能度 (availability)，保守容易度 (serviceability) のための OS の機能の強化が続けられ，それにとくに人間工学的面からの考慮がより払われるべきであろう．

そして，大きなシステムしか使えなかった機能・能力が次第に小さいシステムでも実現され，小規模システムのユーザーを含めて，より多くの人々が OS の恩恵に浴することになろう．

ユーザーの需要に応じた OS の設計目標，概念，手法，その開発の技術と管理，さらに開発されたものの効果的利用，保守サービスなどに一層の前進を期待しつつ，この「OS の展望」を終わることにしたい．

付 記

講演後に本誌編集幹事の方から，とくにご要請があったので，講演の内容の中で一般にあまり知られていない二，三の事がらの参考文献について，下記を追加させていただくことにする．

マルチタスクのオペレーティング・システムにおけるタスクのディスパッティングの優先度の動的な変更については，

"Dynamic Calculation of Dispatching Priorities under OS/360 MVT" by B. S. Marshall, Datamation, August, 1969.

"A Heuristic Approach to Task Dispatching" by K. D. Ryder, IBM Systems Journal, Vol. 9, No. 3, 1970.

に報告されている．

PLAN (Problem Language Analyzer) は IBM の 1130, DOS/360, OS/360 に対して開発され，利用されているが，1130 PLAN については，

"Problem Language Analyzer (PLAN) (H 20-0490)" IBM, January, 1969.

に概略が記されている．ICES (Integrated Civil Engineering System) は，MIT の土木工学科で システム/360 用に開発されたシステムであるが，その文献としては，

"An Integrated Computer System for Engineering Problem Solving" by D. Roos, Proc. FJCC, Vol. 27, pt. 2 (1965).

"ICES Systems Design (2nd ed., revised)" by D. Roos, M. I. T. Press.

などがある．

システム管理機能 (System Management Facilities) は，計算機システムの内部使用に関する情報を収集・記録するためのソフトウェアの機能で，OS/360 に開発され，その機能や利用法などは

"System/360 OS Systems Management Facilities (C 28-6712)" IBM, July 1970.

に記述されている．

比較的小規模な汎用の OS の下にタイム・シェアリングを行なう例としては，システム /360 のモデル 25 以上で，DOS なら 40K の区画で 10 内至 12 の端末が使える ITF (Interactive Terminal Facility) がある．これの概略については，

"DOS/360 Interactive Terminal Facility Program Product Design Objectives (C 28-6821) July, 1970"

を参照されたい．