第Ⅰ章　対話型処理システム

第Ⅰ章　対話型処理システム

学習目標

1．現在コンピュータ処理形態の主流が対話型処理となっている状況をアプリケーション例を引用して認識する。これにより対話型処理の特徴を捉える。また、対話型処理における応答時間がいかに利用者の生産性と大きな関係があるかを理解させる。

2．対話型処理システムの実現構造

対話型処理を実現するために、人とコンピュータとの意思疎通をどう構造化するか、また、その処理を実現するためにコンピュータプログラムにどのような仕掛けが必要となるかを理解させる。

3．ユーザインタフェース

ユーザインタフェース技術を概観し、現在の操作性の水準がどこまで達しているかを理解させる。

全体概要

計算機処理は1970年代中葉までの汎用コンピュータによるバッチ処理主体の計算処理と入出力全盛の時代から、汎用コンピュータ上でTSS形態による対話的なコンピュータ利用が可能となることにより次第に対話型処理が幅をきかせ始めた。1980年代に入りミニコンピュータの急速な普及、パソコンの導入、1980年代半ばにはワークステーションの利用へと対話型のコンピュータ利用が圧倒的に増えていった。

近年、バッチ処理は影に隠れながら相変わらず根強く生き続けてはいるものの、コンピュータ利用者の間では死語に近い状態にある。本章では、ことさら対話型処理という言葉を強調するのではなく、むしろ現在のコンピュータの自然な使い方は対話が基本となるという観点でその特徴を解説する。

内容のあらまし

節　・　項	内　容
1．対話型処理の特徴	
（1）対話型処理	文字主体の対話型処理 図形主体の対話型処理 画像主体の対話型処理 グラフ主体の対話型処理
（2）対話型処理システムの構造	汎用コンピュータ端末での対話 パソコンでの対話 ワークステーションでの対話 一般論としての対話処理のメカニズム
（3）対話型処理とバッチ処理	
（4）応答時間の重要性	ターンアラウンドタイム レスポンスタイム システム応答時間 ユーザ応答時間
2．対話型処理システムのオペレーティングシステム	
（1）対話型処理システムの実行制御構造	コンピュータと人間の対話 イベントの発生からみた対話型処理
（2）1台のコンピュータによる複数の同時作業	
（3）オペレーティングシステムの役割	主記憶装置の管理
（4）プログラムの実行管理	プログラムという概念
（5）対話型オペレーティングシステムの構造	
3．ユーザインタフェース	
（1）ユーザインタフェースの意義	
（2）典型的なウィンドウシステム	
（3）ユーザインタフェースシステムの例	
4．主要用語	

1．対話型処理の特徴

（1） 対話型処理

ワープロ、パソコン、ワークステーション、汎用コンピュータの端末のディスプレーを見ながらキーボードやマウスを使って利用者自らがプログラムの動作をナビゲーションして、所望の計算処理結果を得る方法である。

ディスプレー上では、文字、図形、画像、グラフなど種々の形式で情報が表現され、またさらには音なども発することが可能である。この型の処理においては複雑で大量な計算処理ではなく、むしろそれは別の局面で処理が終わっており、計算の途中結果や最終結果をみながら利用者が何を考えるか、どのような意思決定を行うかのプロセスが中心となる。

以下で、技術計算における対話例を見てみよう。

対話の例１：

コンピュータ：「何をお望みですか？」
利　用　者：「構造解析用のソフトウェア“STANAL”を実行したい。」
コンピュータ：「私は＊STANAL＊です。解析対象のファイル名を教えて下さい。」
利　用　者：「“××エンジン外形”を指定します。」
コンピュータ：「解析条件を教えて下さい。」
利　用　者：「［X1，Y1，Z1］と［X2，Y2，Z2］で表される領域について形状を計算して下さい。」
コンピュータ：「はい計算しましょう。」～～～「計算結果をお見せします。」「結果を印刷しますか？」
利　用　者：「ZZZPRINTERに印刷して下さい。」
コンピュータ：「他のケースを解析しますか？」
利　用　者：「いいえ、一旦終了します。」
コンピュータ：「何をお望みですか？」

対話の例２：

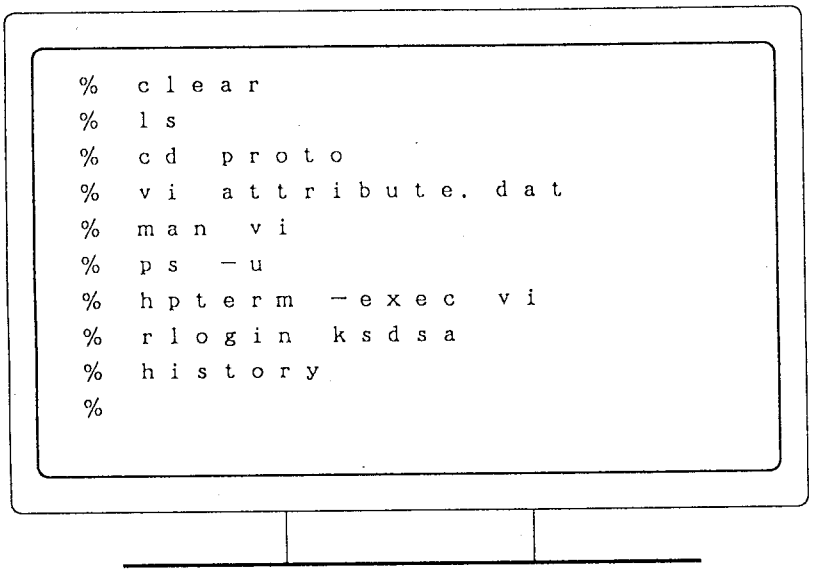
コンピュータ：「何をお望みですか？」
利　用　者：「線形計画ソフトウェア“MPS”を実行したい。」
コンピュータ：「＊MPS＊の実行開始です。対象のファイル名を教えて下さい。」
利　用　者：「“輸送量の最小コスト計画”を指定します。」
コンピュータ：「標準的な計算手続きは～～です。変更しますか？」
利　用　者：「とりあえず、コストが～～百万円になるまで計算を進めて下さい。」
コンピュータ：「途中まで計算しました。しかし、～～百万円までコストを下げられません。どうしましょう？」
利　用　者：「う～ん。ちょっとXXの係数値をYY変えて、最初から計算し直して下さい。」
コンピュータ：「計算できました。～～百万円よりやや安くなりました。この後どうしますか？」
利　用　者：「最低コストになるまで計算を続けて下さい。」
コンピュータ：「計算が終わりました。他の計算を行いますか？」
利　用　者：「いいえ、一旦終了します。」
コンピュータ：「何をお望みですか？」

対話型処理の例

［1］ 文字主体の対話型処理

文字を主体とした対話シーケンスを例示する（図表Ｉ－１）。

図表Ｉ－1　文字主体の対話型処理

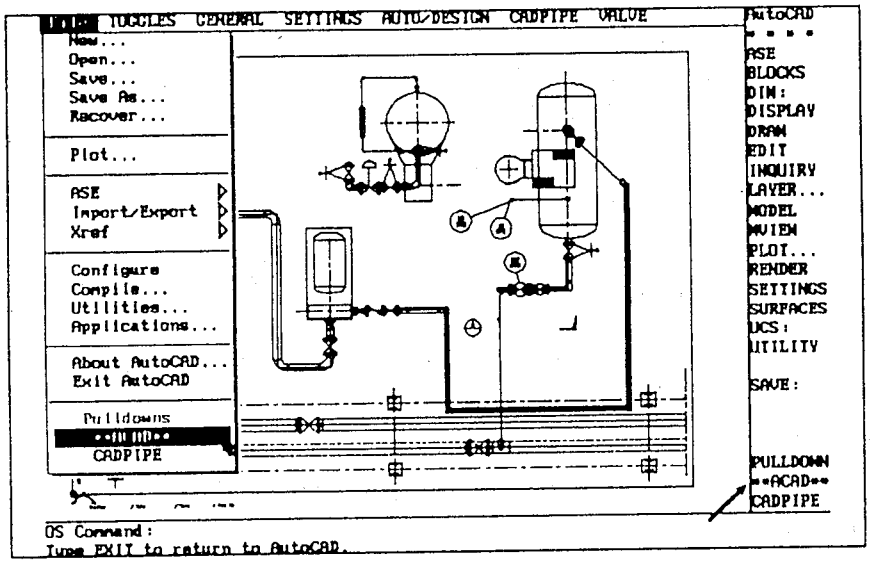


［2］ 図形主体の対話型処理

配管系のCADソフトによりプラントの配管図形を作る対話を例示する（図表Ｉ－２）。

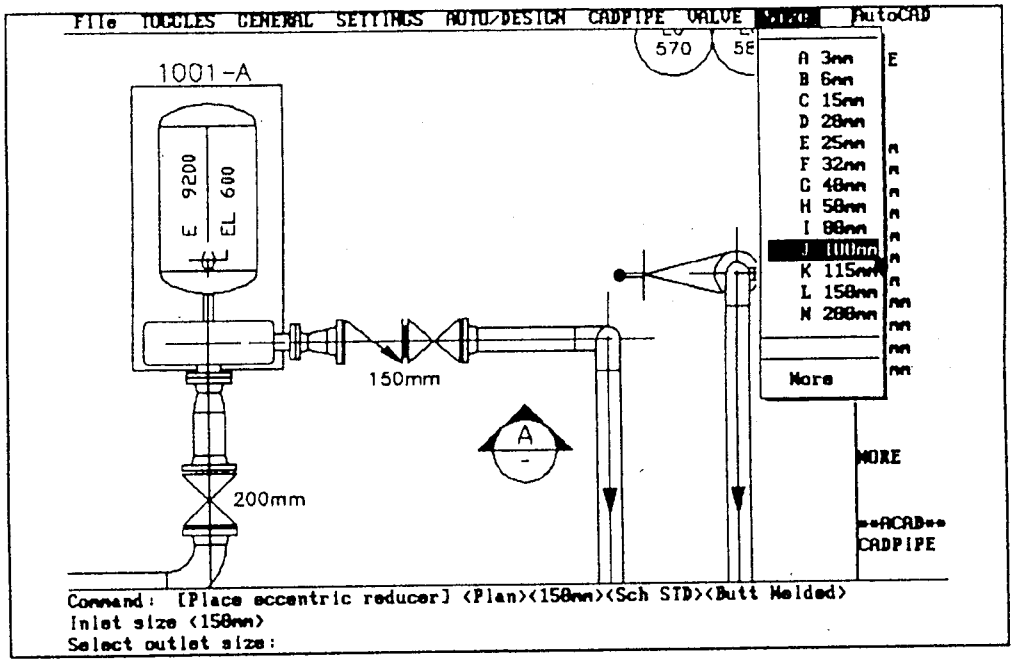
下例では、メニュー“ファイル”を選択すると、プルダウンメニューが出てくる。そこで、下から２番目の＊＊ACAD＊＊を選択すると、メニューバーの内容が次頁のように変わる。

図表Ｉ－2　図形主体の対話型処理1



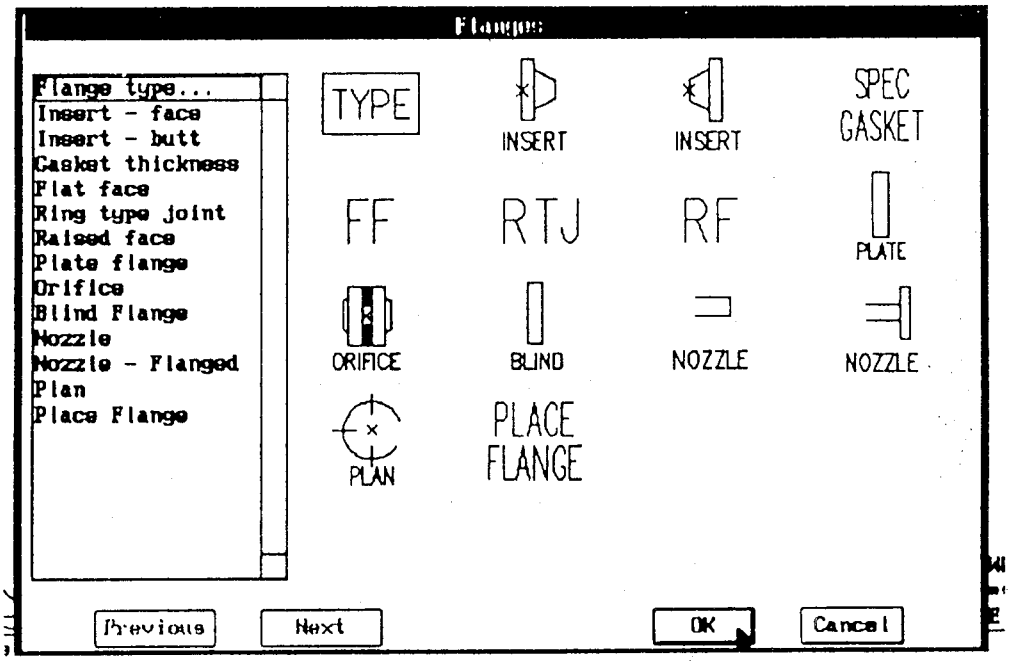
引き続き、レデューサーを置くことを想定する。この時アウトレットのサイズをどうするかを訊ねられる。ポップダウンメニューが表示され、どれかを選ぶ（図表Ｉ－３）。

図表Ｉ－３　図形主体の対話型処理２



レデューサーを置くときの繋ぎ口の形（フランジ）に関して、アイコンメニューが提示され、どれにするかをマウスで選ぶ（図表Ｉ－４）。

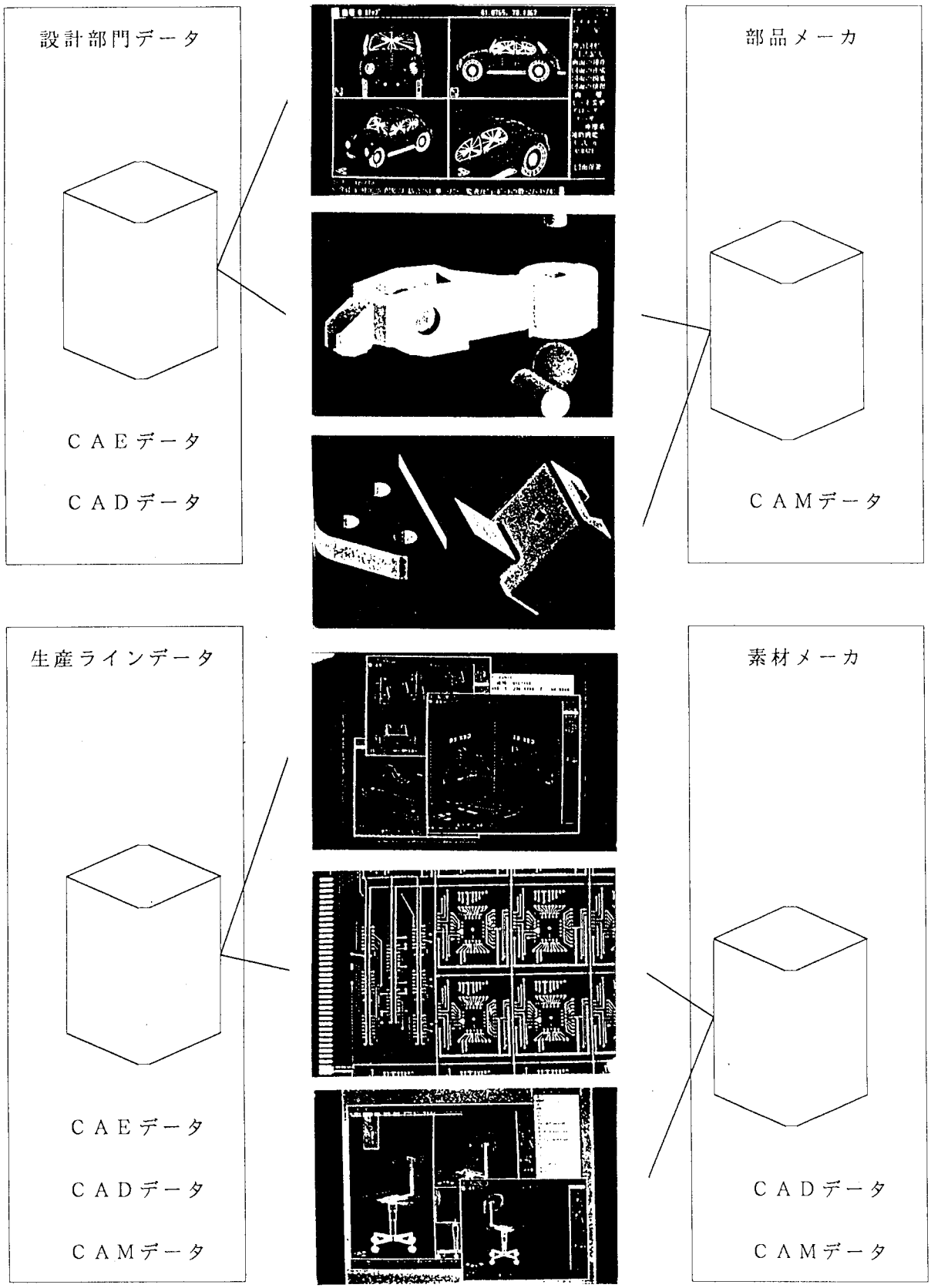
図表Ｉ－４　図形主体の対話型処理３



［３］　設計主体の対話型処理

CADシステムにおける対話（図表Ｉ－５）

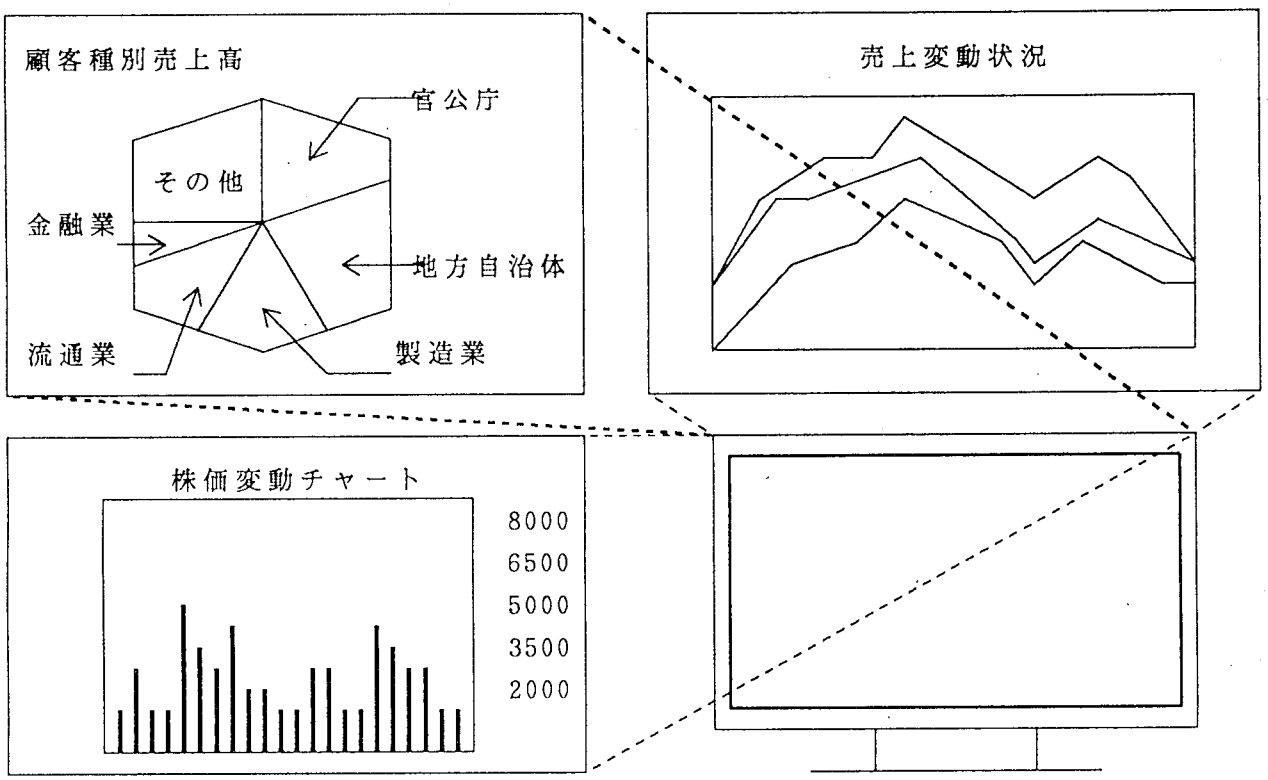
図表Ｉ－５　設計主体の対話型処理



［4］ グラフ主体の対話型処理

経営情報など意思決定支援システムなどの処理例で説明する(図表Ⅰ－6)。

図表Ⅰ－6　グラフ主体の対話型処理



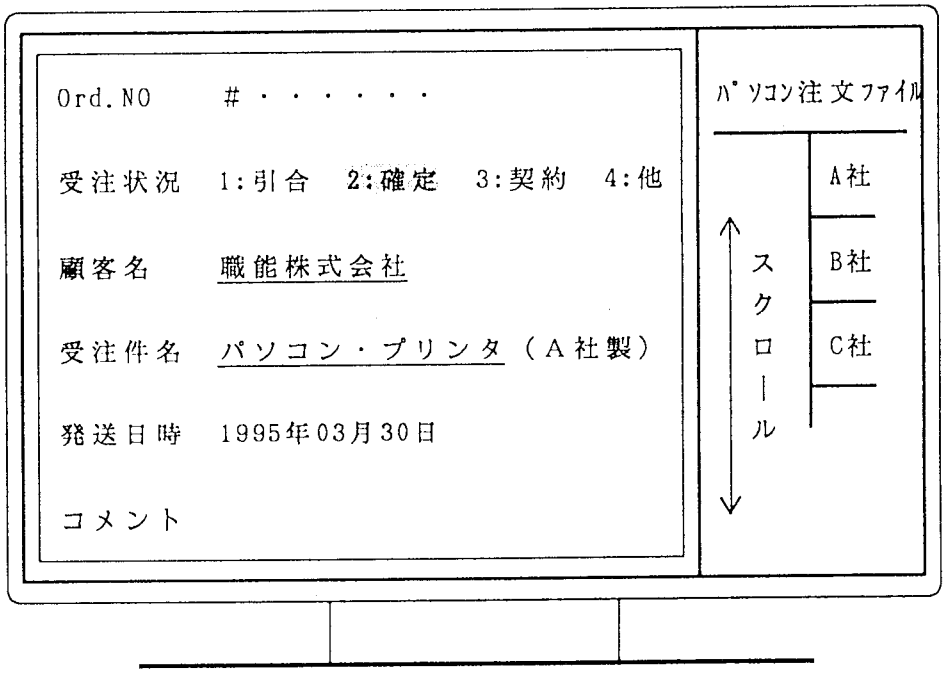
（2） 対話型処理システムの構造

一言で対話型処理と呼んでしまっているが、それを実現する構造はいろいろある。ここでは代表的な対話型利用例を引用してコンピュータサイドの構造を説明する。

［1］ 汎用コンピュータの端末での対話

フルスクリーン型の端末を利用しての伝票処理の例（図表Ⅰ－7）

図表Ⅰ－7　端末での対話



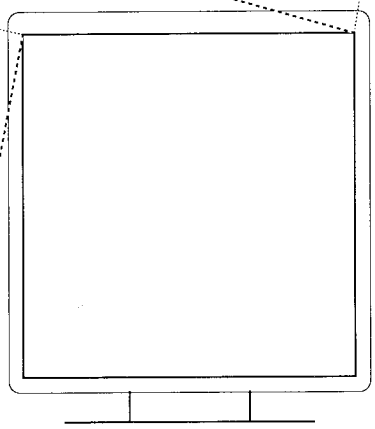
［2］ パソコンでの対話

表計算プログラムを使うOA業務処理の例（図表Ⅰ－8）

図表Ⅰ－8　パソコンでの対話

請負番号	顧客名	製品名	金額	既入金額	既材料費	人件費	経費計
XYZ51255	ABCカンパニー	Yタイプシリーズ	25,500	10,000	8,000	3,000	11,000
XYZ51423	ＦＯＯ㈱	Nタイプシリーズ	42,000	15,000	20,000	9,000	29,000
XYZ61062	白用舎	Nタイプシリーズ	91,500	23,000	45,000	16,000	61,000
合計	N顧客		233,000	100,500	100,500	56,500	157,000

	上期	下期	年間
売上高	2,386,623	3,103,452	5,490,075
人件費	795,541	1,034,484	1,830,025
共通経費	60,000	72,500	132,500
小計	855,541	1,106,984	1,962,525
材料費	923,236	1,258,295	2,181,531
期首仕掛品	642,310	598,425	642,310
期末仕掛品	-598,425	-525,799	-525,799
販管費	356,390	356,390	712,780
営業利益	207,571	309,157	516,728
営業外費用	1,150	1,200	2,350
経常利益	206,421	307,957	514,378

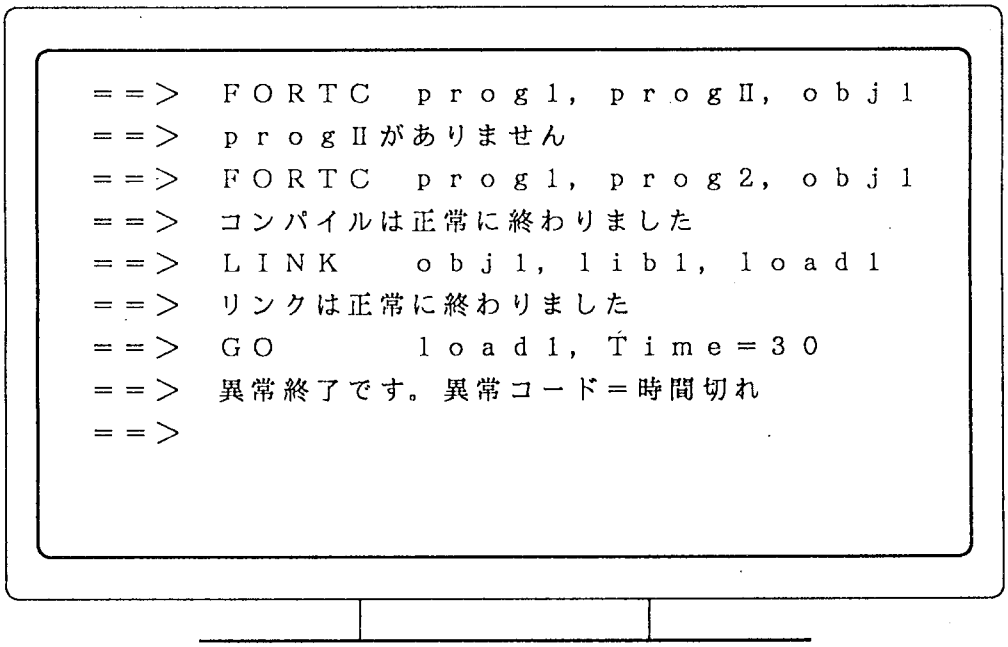


［3］ 一般論としての対話処理のメカニズム

ハードウェア、オペレーティングシステム、アプリケーションプログラムおよび利用者の入力操作による対話処理の基本構造を見る。三つのタイプ。

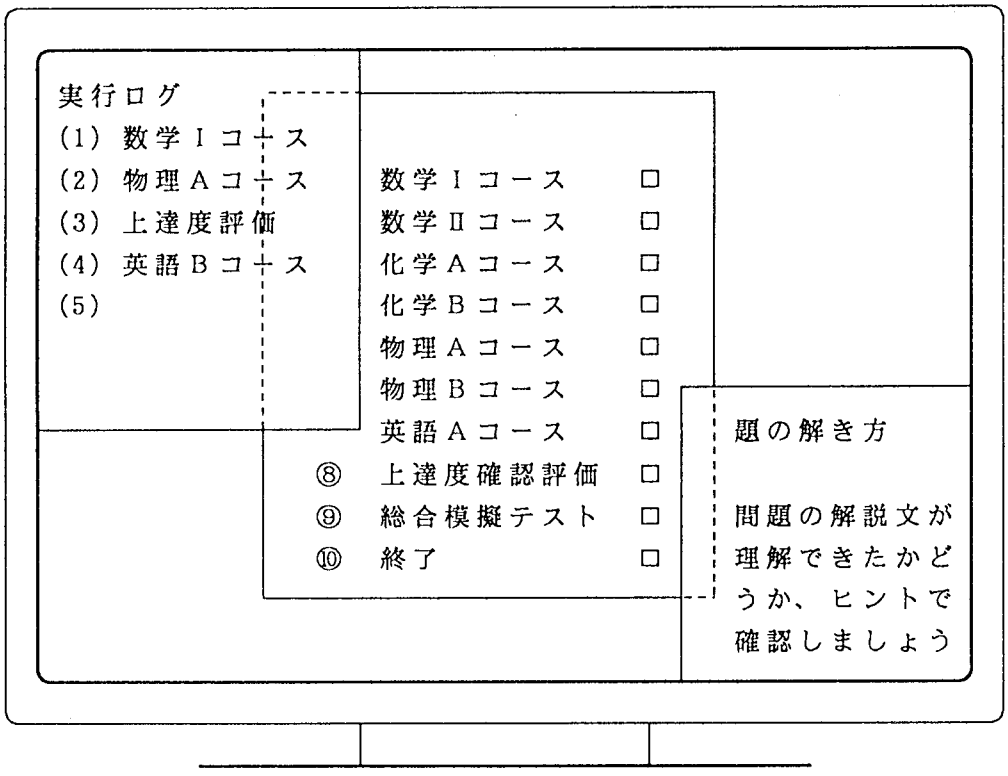
① 原始的な対話処理－オペレーティングシステムやコマンドインタプリタのサポートによる対話（図表Ⅰ－9）

図表Ⅰ－9　原始的な対話処理



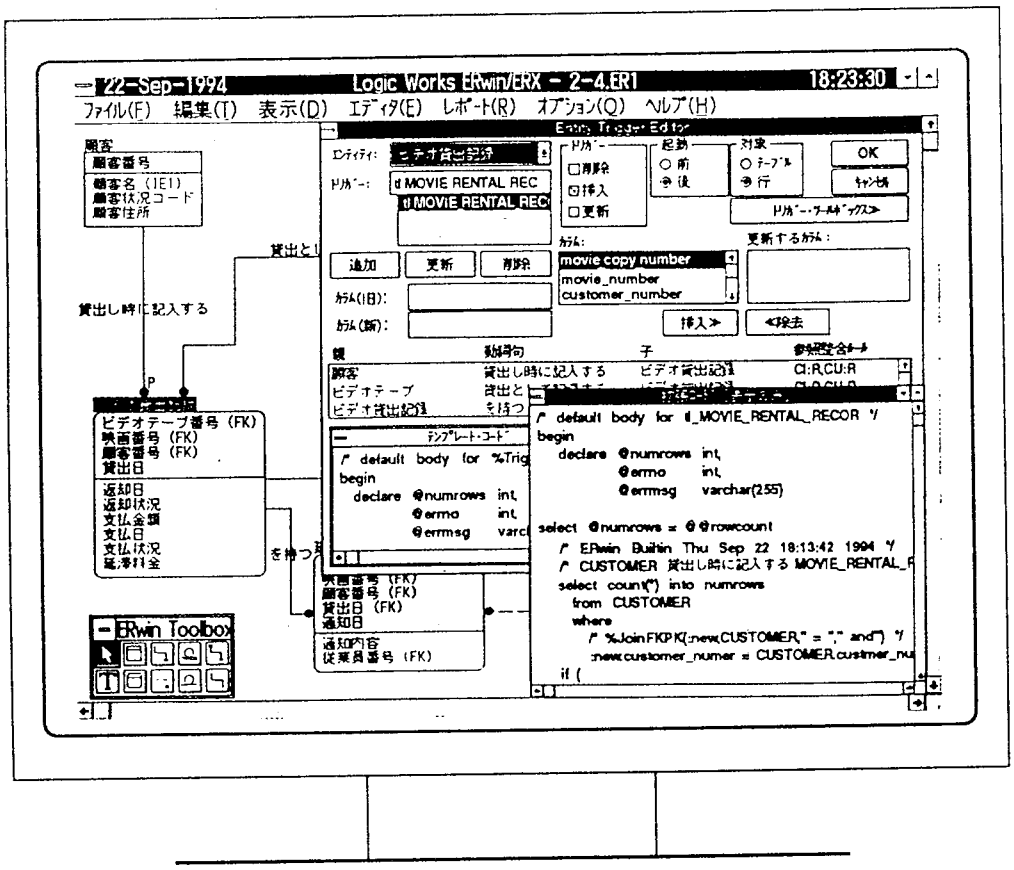
② やや進歩した対話処理－アプリケーションプログラム自らが対話の主導権をとる方式（図表Ⅰ－10）

図表Ⅰ－10　やや進歩した対話処理



［4］ マルチウィンドウを用いた例（図表Ｉ－11）

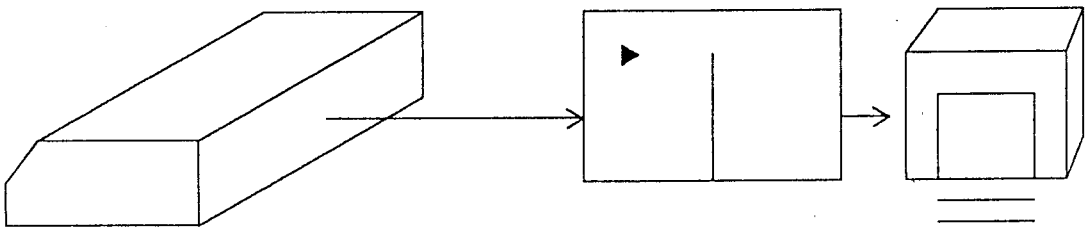
図表Ｉ－11　マルチウィンドウを用いた例



（３） 対話型処理とバッチ処理

もはやバッチ処理が全く不必要になったというわけではなく、どのような要求に対してバッチ処理が必要となるかを対話型処理との対比で説明する。バッチ処理型のプログラムに関してはその実行が始まったら通常人との対話が入ることはない。これはバッチ処理の大きな特徴である。

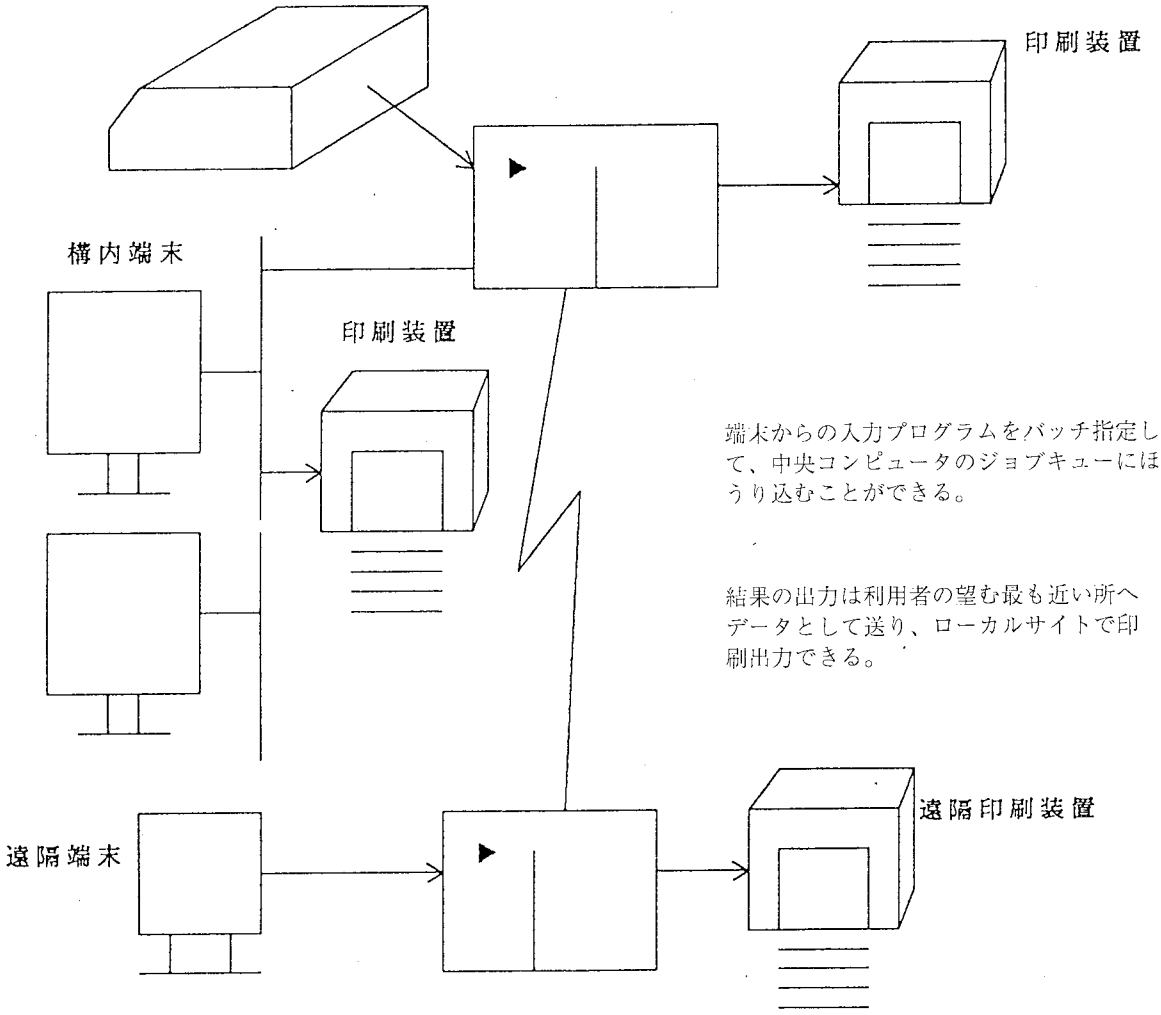
図表Ｉ－12　汎用コンピュータ全盛期のバッチ処理



ジョブは周辺装置のカードリーダから入力

結果出力は印刷装置へ

図表Ⅰ－13　対話型処理時代のバッチ処理



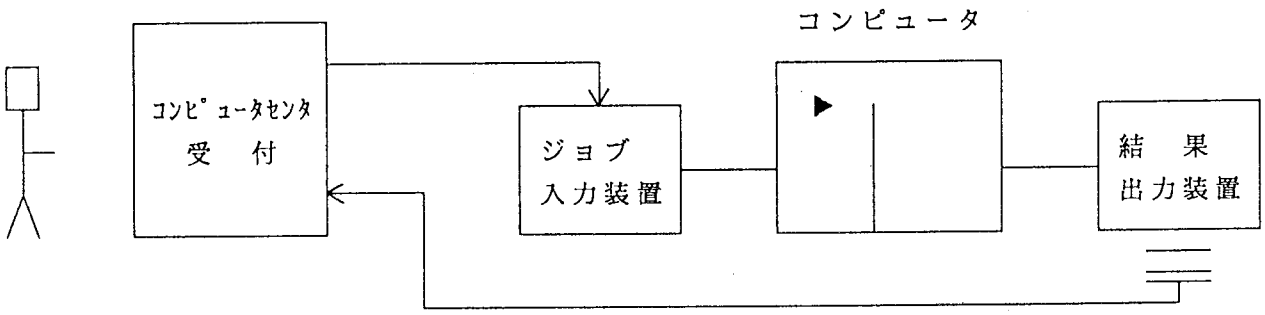
（4）　応答時間の重要性

対話型処理こそが近代コンピュータ利用の要となったとはいえ、大量計算処理にそれを用いても効果はでない。パソコンは、低価格でしかもユーザインタフェースに優れているとはいえ大積みのソフトウェアを動かそうとしても応答性がすこぶる悪ければ意味がない。

ここでコンピュータ利用と利用者の生産性との関連について触れてみよう。

①　ターンアラウンドタイム

この概念は、バッチ処理において使われる応答を示すものであり、利用者がプログラムとデータをまとめてコンピュータに入力し、計算処理が終了して結果を受け取るまでの総時間をターンアラウンドタイム（turn around time）という。このプロセスは、バッチ処理ジョブをセットアップして（プログラムコード、データおよびJCLの作成・修正）コンピュータセンターの受付にジョブ処理依頼を行う。受付係はオペレータに当該ジョブを渡すと、オペレータは投入ルールにそってジョブ入力を行う。計算処理が行われて結果がプリンタ出力されると、オペレータは各結果を分類して入力ジョブとの対応を付けて受付係に返す。受付係から連絡を受けて結果を受け取る。という流れとなり、この間コンピュータのジョブの混み具合や受付処理の迅速性によって応答が変わってくる。受付とオペレータとの物理的な距離が短ければその分待ち時間も少なくなる。



② 応答時間（レスポンスタイム）

対話型処理において、利用者が対話処理端末からコンピュータに指示を送り、その結果がディスプレー上に戻されるまでの時間。結果を得るまでの総時間という点ではバッチ処理におけるターンアラウンドタイムに相当する。応答時間を厳密に分析すると、システム応答時間とユーザ応答時間とに分けることができる。

a．システム応答時間

システムオーバヘッド（利用者の要求に対する応答を行うためにシステムプログラムが所定の手続きを行うに要する負荷）として捉えられる。

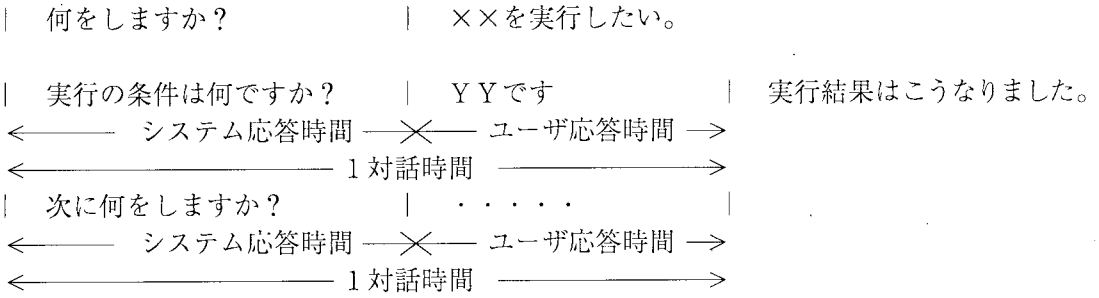
例えば、ある文字列あるいは図形に関する編集処理を行っているとする。ここで利用者が、

・ ある単語を指定してそれを有する文字列の行を探す場合、システムは当該単語を含む行を探して存在すれば該当行の内容を表示するものとする。この場合、そのような行が見つかった時点でそれを表示するまで、また、さらにそれを含む行を見たければ探索を続行し、次の該当行が出てくるまで利用者は待つことになる。このようなシステム側が仕事をしている時間はシステム応答時間に相当する。

勿論、このような時間数は利用者の待ちの感覚をいらだたせない程度の速さは必要となるこの応答性が利用者の対話作業における生産性に直結するからである。この生産性は利用者の思考過程にタイムリーに連携することで上がるものと判断される。

b．ユーザ応答時間

利用者がコンピュータを対話利用しているときに、システムが利用者の要求する機能を実行し終わるまでの時間をシステム応答時間と呼ぶことに対して、実行結果を見て利用者が次に何を要求するかを知らせるまでの時間をユーザ応答時間と呼ぶ。

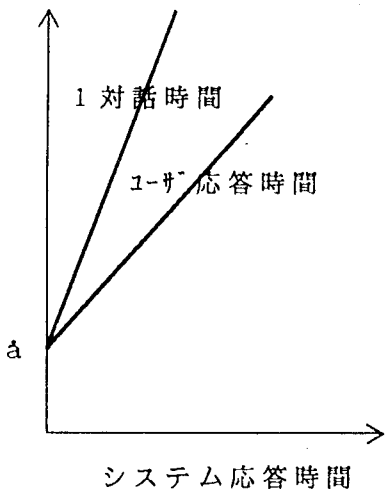


ユーザ応答時間そのものは利用者自身の思考からくる速度であり、これが速いとか遅いとかいう評価をすることは個人差があり適切ではないが、システム応答時間が遅いと感じるシステムでは当然利用者の生産性に影響し始める。

c．ユーザの生産性

S.J.Boiesの研究によると、ユーザの生産性は、応答の速さ、使い勝手の良さに大きく左右される。使い勝手にはオペレーティングシステムにおける基本的な対話処理サポート性、GUI、アプリケーションの操作インタフェースなどからなる。どれか一つでも欠陥があれば、生産性を落とすことになる。

ア．Boiesの現象



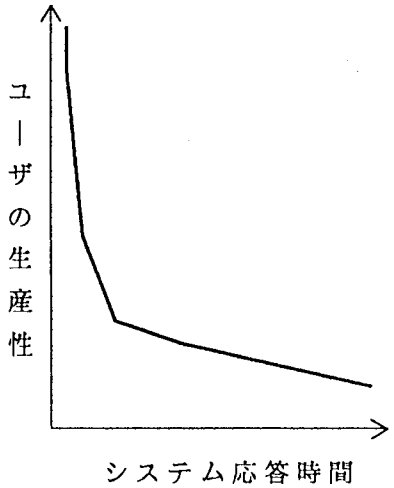
1対話時間＝ユーザ応答時間＋システム応答時間

ユーザ応答時間＝システム応答時間＋a

したがって、
1対話時間＝2×システム応答時間＋a
となる。

出典：第二種共通テキスト
情報処理システム
（(財）日本情報処理開発協会

イ．システム応答時間とユーザの生産性



ユーザの生産性は、1単位時間に反比例するため、システム応答時間との関係は左図のようになる。

ユーザの生産性は、システム応答の速さだけでは計れのない。操作性（使いがって）の良しあしが大きく影響する。
いわゆるユーザインタフェース技術の利用者への支援がどの程度効き目があるかが問題となる。
システム応答だけ速くても、ユーザ応答環境において労働集約的な部分が多ければ、対話のレベルにギャップが発生して思考過程を助ける流れとならない。

出典：第二種共通テキスト
情報処理システム（（財）日本情報処理開発協会）

対話型のアプリケーションプログラムに関しても、GUIツールをスタイルガイドなどに準拠して適切な使用法を考慮して操作環境を作る必要がある。

もう一点重要なことは、同じ利用者であっても初心者と熟練者とでは当然ながら同一に環境に対して異なる生産性を示すことである。初心者は、機器の操作に不慣れであったり、ソフトウェアの操作仕様や機能仕様に慣れがなく、操作に手間取る時間が多く、環境への不満はなかなか出てこない。

一方、熟練者にとっては、応答の速さや操作性の一貫性だけでなく、さらに多くの利便性を求める。ある意味でまどろっこしい部分が残される。この辺は、ヒューマンインタフェースの技術的な課題として、今後研究開発の成果が期待されるところである。

2．対話型処理システムのオペレーティングシステム

（1） 対話型処理システムの実行制御構造

コンピュータとの対話は、基本的には人間対人間の会話のメカニズムと同じである。しかし、いくつかの点では著しく異なる。例えば、人同士の場合、一方が喋りまくり他方が相づちを打つだけ、あるいは互いに主張し合うだけ、また、お互いに感激し喜び合う等々。

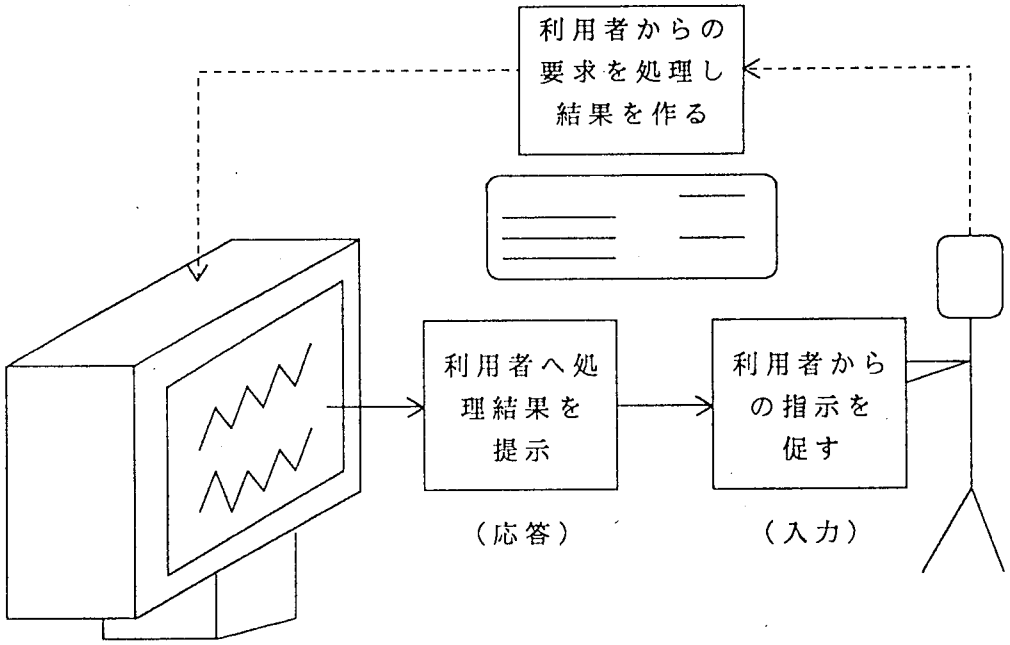
人とコンピュータとの対話では、人はできるだけ簡単かつ少ない入力操作で情報を与えたい。コンピュータにとってはそうしてはあげたいが、あうんの呼吸とか以心伝心あるいは言葉足らずでは意味を理解することは不可能であるし、またそのようなコンピュータの出現を望んでもまだずっと遠い将来の夢であろう。

基本的にコンピュータは、構造的な対話と形式的な表現とでしか人の意図を受け取ることができない。また、そのような対話を可能とするためにコンピュータ側にはしかけが必要となる。これを実現する手だてとしてコンピュータのオペレーティングシステム、ユーティリティ、ユーザインタフェースプログラムなどがあり、それらのうちのいずれかもしくは組み合わせで実現されている。

対話は、ごく簡単なやりとりから非常に複雑なやりとりまでが考えられ、メディアの種類が増えるほど人間にとってはより高度なお願いをコンピュータに要求することができるようになる。当然コンピュータにとってはそれを可能とするためにハードウェアの技術革新やソフトウェアの高機能化を果たさなければならなくなる。

コンピュータと利用者の対話構造は、下図のように最初コンピュータが利用者からの指示（入力）を待つ。この間コンピュータは待ち状態となる。指示の投入があると、待ち状態が解除されて所望の計算処理を行い、応答（結果の出力）を出す。その後再度指示の待ち状態にはいる。

図表Ⅰ－14　対話型処理の実行制御構造



コンピュータが利用者からの指示を待つということは、オペレーティングシステムもしくはその制御下で動作しているアプリケーションプログラムがアイドルの状態にあることを意味している。この時点では、利用者はこれから何を実行しようかあるいは直前の計算結果を見て次に何をすべきかを考えていることになる。

ところで、コンピュータへ指示する操作としてはキーボードを使って文字（単語、数値、単文など）を入力する、あるいはマウスやトラックボールを使ってボタンを押すなどの方法が考えられる。電子ペンを用いればキーボードやマウスの代替にもなろう。コンピュータの方はこの入力を待っているわけである。つまり外部からの圧力（きっかけ）によりコンピュータは次の何をすべきかを判断している。

この“きっかけ”を事象（イベント）という。オペレーティングシステムにとっては事象は利用者の入力の一種類だけではなく他にも、接続されている外部機器からの種々のデータの入力あるいは外部装置への出力の完了などが存在する。オペレーティングシステムにしろその配下で動くアプリケーションプログラムにしろ、事象の発生を待ち、発生したら次の仕事にとりかかるといったプログラムの流れになっている。このような処理を一般にイベント駆動（Event driven）という。対話型処理の実行制御はイベント駆動である。

（2）　1台のコンピュータによる複数の同時作業

汎用コンピュータは、同時に複数人で種々の処理を行えるよう設計されている。当然大きな主メモリ、外部記憶装置量、計算能力（CPU性能）が求められる。ワークステーションも汎用コンピュータに比較すれば比べるべくもなくパワーは劣るが、利用者個々人からみれば高性能であり、1台を複数人で利用することができる。

パソコンに関しては、ここ1～2年で相当高速なチップ能力を持つマシンが出始めているが、基本的には個人が占有して使う程度である。このようにコンピュータには複数人が同時に利用できるものと1人だけでの利用とに大別され、オペレーティングシステムの役割は単一利用者か複数利用者かで大きく変わってくる。オペレーティングシステムにとっては単一利用よりも複数利用の方が仕事の種類や量がずっと多いことは容易に想像されよう。一般にはシングルユーザ

OS、マルチユーザOSという呼び方をしている。

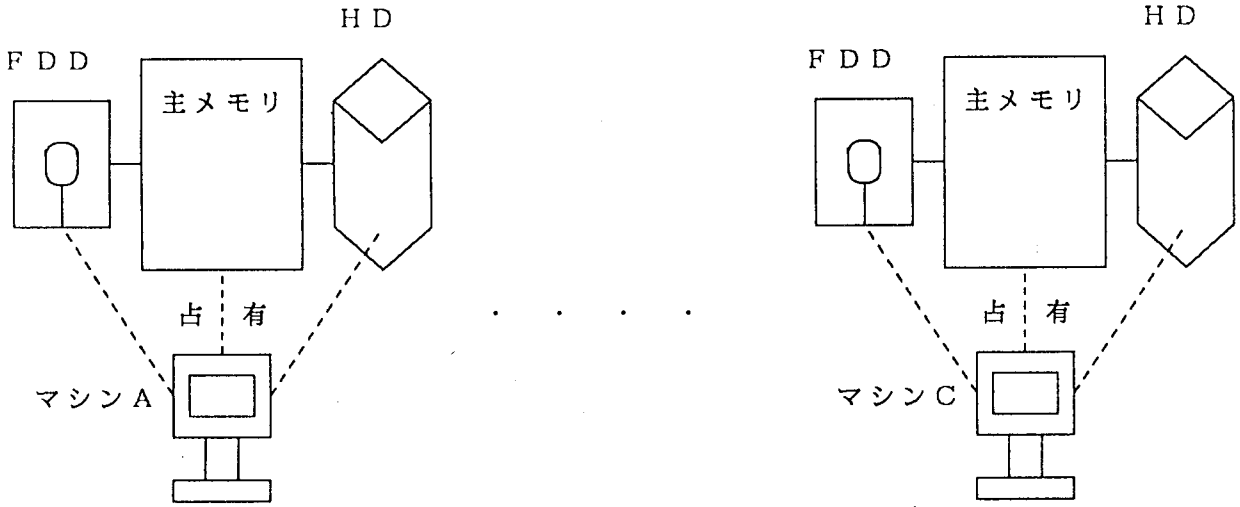
シングルユーザとは、１台のマシンを一人で使うことを意味するが、一人で使う場合であっても同時に複数の種類の作業をこなそうとする場合が多い。例えば、一度に複数個の文書ファイルを編集したい、あるいは編集をしているときにデータベース化されている情報を参照したいことはごく自然の要求である。この要求を満たすためにオペレーティングシステムは、マルチプログラミングあるいは多重プログラミングという仕掛けを有してなければならない。同時に、実行される各々の仕事がそれぞれ別のプログラムに対応しており、同時に動かすための基本的なOSの機能である。

このマルチプログラミングの機能がなければ、文書の編集最中に関連情報をデータベースから検索をしたくても、一旦編集プログラムを終了させ、検索プログラムを起動し情報を得たらそれを終わって再び編集プログラムを起動するという面倒な操作で進めなければならない。

（3） オペレーティングシステムの役割

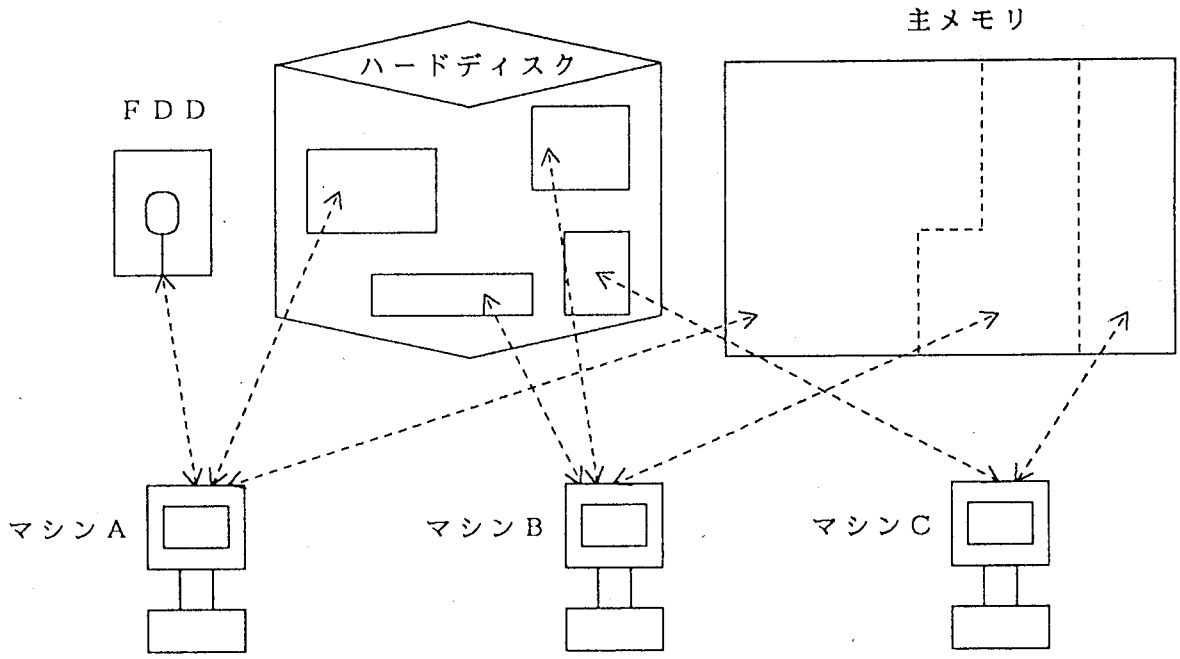
汎用コンピュータでは、マルチユーザ（一度に複数個のバッチ処理プログラムを実行させる、一時に複数個のTSS端末処理を行う、多地点からのオンライン情報に対応して適用業務プログラムを処理するなど）処理を行うために、多重プログラミングの概念をいち早く導入した。対話型処理においてもマルチユーザという使い方は存在するが、現実にはむしろ、マルチウィンドウ処理を行うために多重プログラミング機能を持ち込んでいることになる。

多重プログラミングにおいては、コンピュータシステムの持つ全てのハードウェアおよびソフトウェア資源を一括管理する必要がある。これをシステム資源管理機能と呼んでいる。単一プログラミングにおいてはシステム資源の競合は存在しない。



多重プログラミングにおいては、オペレーティングシステムは同時に利用する各利用者があたかもコンピュータを一人で占有しているごとく見せかける必要がある。

図表Ⅰ－15　多重プログラミングの仕掛け



マルチユーザの実現は、オペレーティングシステムにとっての多重プログラミングの機能とシステム資源に対する同時使用要求をかなえるためのシステム資源管理機能により可能となる。ここでいうシステム資源とは、CPU、主メモリ、外部メモリ、ファイル、ソフトウェアなどを対象としている。

① 主記憶装置の管理

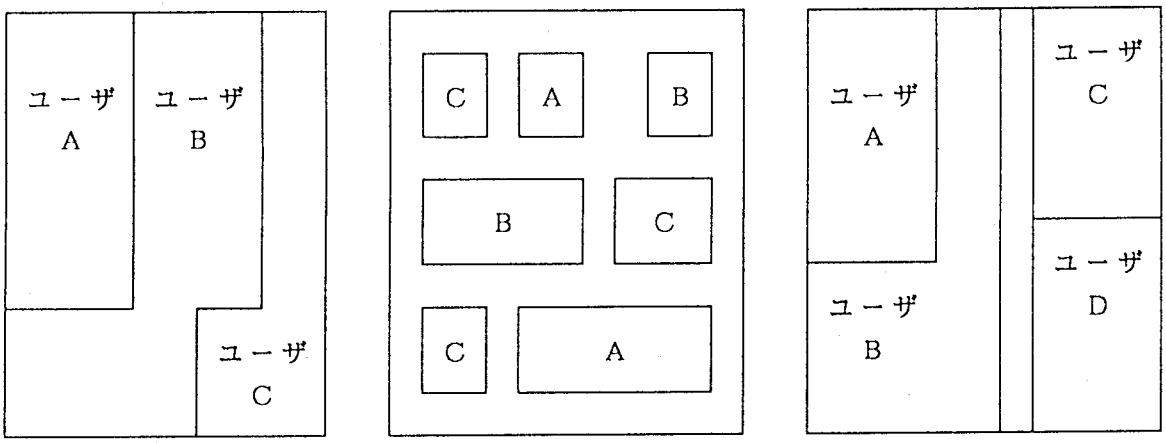
同時に動作する各ユーザ毎に必要な量の主メモリを割り当てる。ユーザが主メモリの残量よりも大きな量を要求したら、いずれかのユーザがメモリ割当てを待たなければならない。ユーザの仕事が完了すれば、主メモリは解放されるので、メモリ待ちのユーザはその時点で割当てが行われる。

この割当て方式には何通りかが実用化されてきた。例えば、

- ユーザの要求する必要量を空きメモリから連続して割り当てる。
- ユーザの要求する必要量を小さな破片空きメモリをかきあつめて割り当てる。
- ユーザの要求する必要量を一度に全部割り当てるのではなく、プログラムの実行に最低限必要な量だけを割り当てる。

などの方式がある。これを主記憶の空間分割と呼ぶ。

図表Ⅰ－16　主記憶の分割の例



② CPUの管理

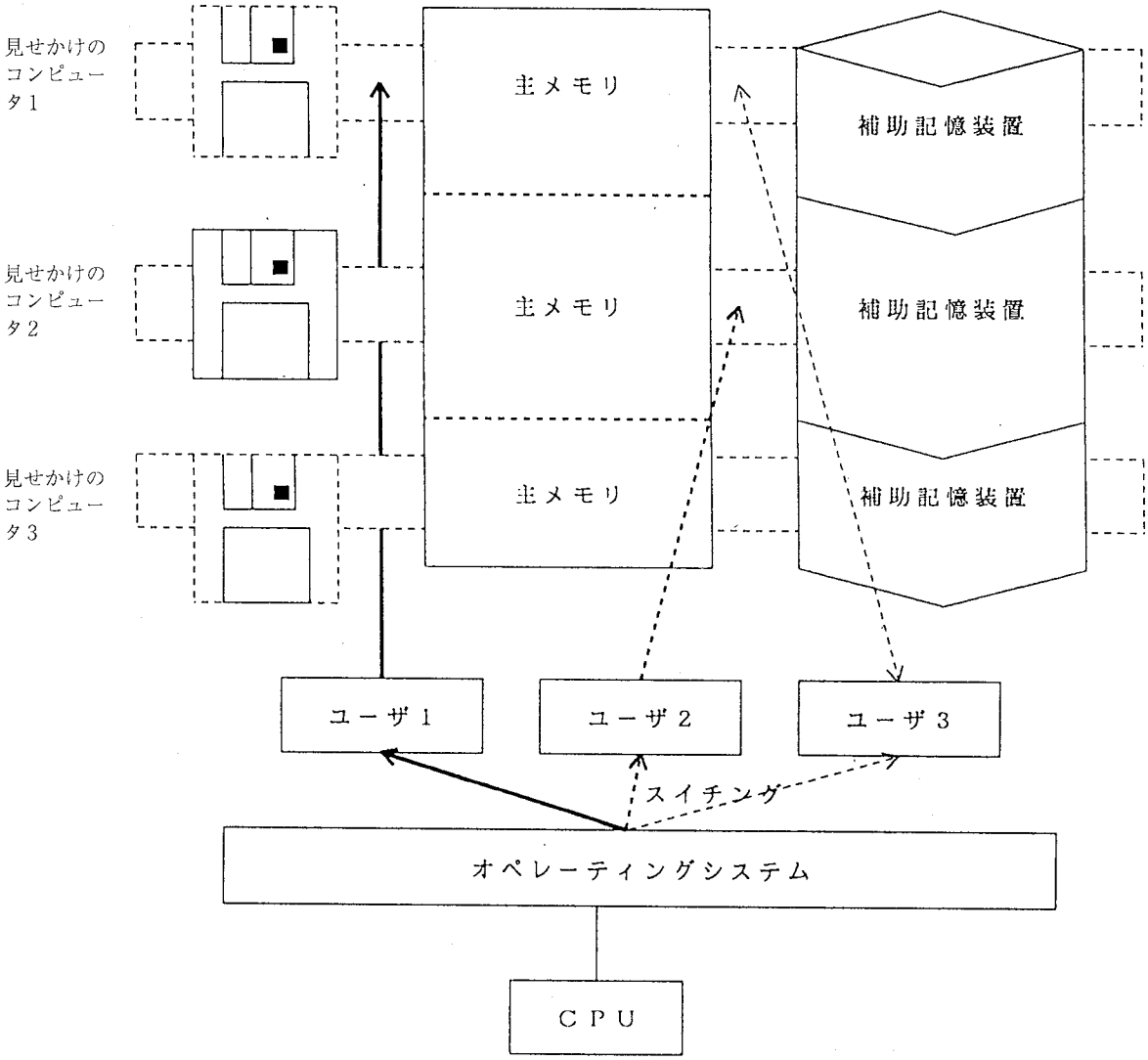
CPUの稼動率を上げるために、また、各ユーザの所望するパフォーマンスを保証するために時分割方式により、CPUの使用権を付与することはオペレーティングシステムにとって非常に重要である。時間をスライシングしてそれぞれの時間刻みにプログラムの実行権を与える方式をとる。

時分割でCPUを使う例として、タイムシェアリングシステムでは、ラウンドロビン方式と呼ばれ、コンピュータに接続されている各TSS端末に対して等間隔でCPUの実行権を与え全端末を巡回する方法がある。TSSでは、コンピュータと利用者との対話が多く、僅かづつCPUが使えれば、自分が独占しているように見える。

また、バッチ処理などにおいて、CPUバウンドなジョブに優先度の高い実行権を与えてしまうと、他のジョブがまったく動けなくなる可能性があり、一度に占有できる時間数を制限して交通整理する方式がある。

オペレーティングシステムは、システム資源の使用状況を管理し、コンピュータを同時使用するユーザに対して平等に同じ様な機能を提供することになるが、これを見せかけのコンピュータと呼ぶこととし、イメージ的には図表Ⅰ－17のようになる。

図表Ⅰ－17　見せかけのコンピュータ



見せかけのコンピュータ
　＝時分割CPU実行権＋割当て主メモリ＋割当て補助記憶装置

一般に、主メモリやディスクなどは見せかけの資源として割り振られるが、デバイス型の装置例えば磁気テープ、フロッピー装置などは、見せかけの管理をせずに、オペレータの人為的な管理により見せかけの資源としている。コンピュータ用語としては、これらの見せかけの資源を仮想記憶、仮想ディスク、仮想コンピュータなどと呼んでいる。

（4）　プログラムの実行管理

多重プログラミングにおいては、見せかけのコンピュータ上でユーザのプログラムが同時に動作することとなるが、オペレーティングシステムの管理方法から見るとプログラムという言葉はいささかあいまいな存在である。一般にはプログラムとは、一種類以上の仕事を行う集まりである。したがって、プログラムをもう少し実行管理を行い易いように細分化する必要がある。

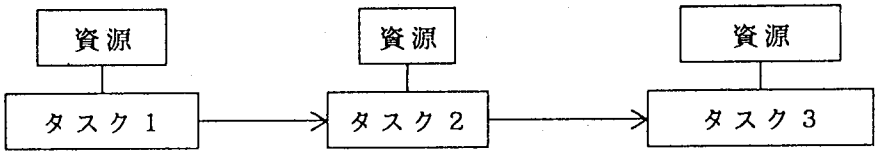
見せかけのコンピュータを提供する対象は厳密にはタスクもしくはプロセスと呼ばれる。オペレーティングシステムがプログラムをタスク（あるいはプロセス）の集まりであるというように

分割する理由を考えてみよう。

例えば、あるコンピュータ言語で書かれたプログラムに関してコンパイル（翻訳）、リンク（連係編集）、ゴー（実行）を行うバッチジョブを想定する。ジョブはJCL、実行プログラム、実行用データを一まとめにしてコンピュータに投入されるが、オペレーティングシステムは、JCL情報を解析して３つのジョブステップに分解する。

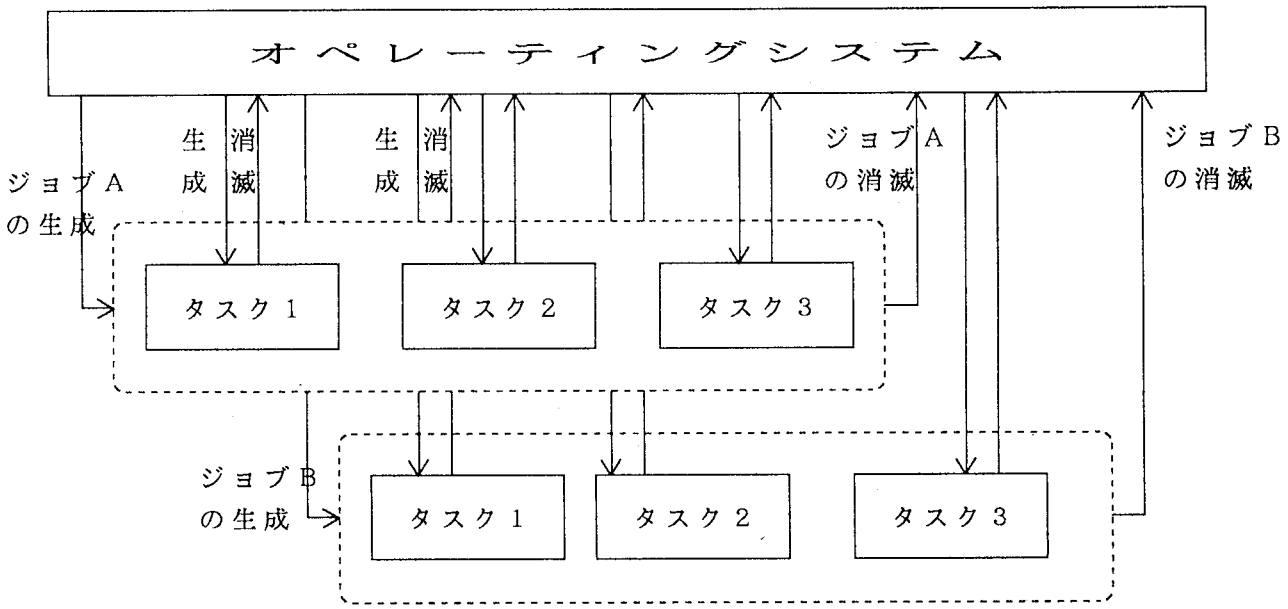
- コンパイルのステップでは、ソースコードが主要入力となり、その他、翻訳を行うために必要な補助記憶装置に作られるファイルおよび出力ファイルとしてのオブジェクトが生成される。実行プログラムはコンパイラであり、コンパイルに必要な主メモリの割当ても必要になる。そのサイズはコンパイラの手続きやデータ領域の大きさにほぼ等しい。
- リンクのステップでは、直前に生成されたオブジェクトや既存のオブジェクトあるいはライブラリを一緒にして一つのロードモジュールを作る。既存のオブジェクトやライブラリには通常諸処の利用者に共通なものが含まれる。オペレーティングシステムが管理する資源としては、リンケージエディタが必要とする主メモリの他、連係編集上必要な中間ファイル、共用ライブラリ（あるいはオブジェクト）ファイル、出力としてのロードモジュールファイルなどがある。
- ゴーのステップでは、直前に生成されたロードモジュールが主メモリにロードされ、先ず静的に必要な主メモリ量が決定される。実行に入ってから実行プログラムが動的にメモリを確保しにいくこともあるが、これもオペレーティングシステムのメモリ管理機能が制御を行う。プログラムの実行においては、種々の入力用、中間用あるいは出力用のファイルが補助記憶装置上に必要となる。CPUの最大使用量なども設定できる。

このように、各ステップでは必要なシステム資源の種類や量が異なる。したがって、オペレーティングシステムが単位時間にできるだけ多くのユーザの要求をかなえるためには、効率の良い資源管理が必要となる。もし、ジョブという単位で資源を割り振っていると常に最大の量を投入しておかなくてはならず、使用効率は低下する。このような状況を背景に、一つのジョブの各ステップは、タスクという管理概念に分割され、オペレーティングシステムによって制御されることとなる。



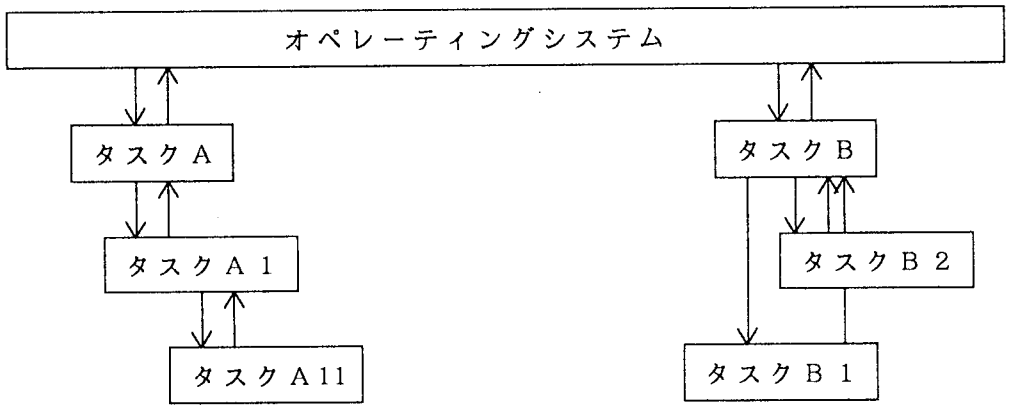
タスクという概念は、オペレーティングシステムによって各々のプログラムを実行するときの管理上の条件を定義するために導入された。このタスクは、ユーザが指定したプログラムの実行に先立って生成され、実行が終了すると消滅される。タスクにはCPUの利用権が付与され、スケジューリングの単位となる。

図表Ⅰ－18　タスクの生成と消滅



図表Ⅰ－18において、ジョブAおよびBのそれぞれのタスクはシリアルに生成され消滅されて行くが、ジョブやタスクがどのような順に終了するかは予測できることではない。したがって、図のような順（左から右へ書いた順）で生成や消滅されると決まっているわけではない。

なお、各ジョブでは、タスクの生成と消滅は必ず逐次行われるように見えるが、これはバッチ処理の特徴からそうなる。実際には、対話型処理やリアルタイム制御処理においては、むしろ並列に行われることの方が多い。勿論、バッチ処理においてもタスクの並列実行は可能である。下図では、タスクAが子タスクA1を、A1がその子タスクA11を、タスクBが子タスクB1と子タスクB2を続けて生成している。



次に、オペレーティングシステムがタスクの実行スケジューリングを行うために内部的にどのようなメカニズムを採用しているかを示す。

これは、CPUの実行権をどのようなタイミングで付与するかに相当する。すなわち、実行権を付与しているタスクが何らかの要因で実行中断もしくは終了したときに他のタスクに実行権を移動するかである。中断の要因として、外部データの入出力を行う、障害が通知される、時間切れとなるなどが考えられる。中断が発生した場合、当該タスクはそれに伴う処理が完了しないかぎりは再びCPUの実行権を与えられることはあり得ない。

このため、各タスクに対してCPUの実行権を得る資格があるかどうかの状態を管理しなけれ

ばならない。大ざっぱには、タスクの状態は下記のいずれかを遷移する。

［1］ 実行状態
［2］ 実行可能状態
［3］ 待機状態

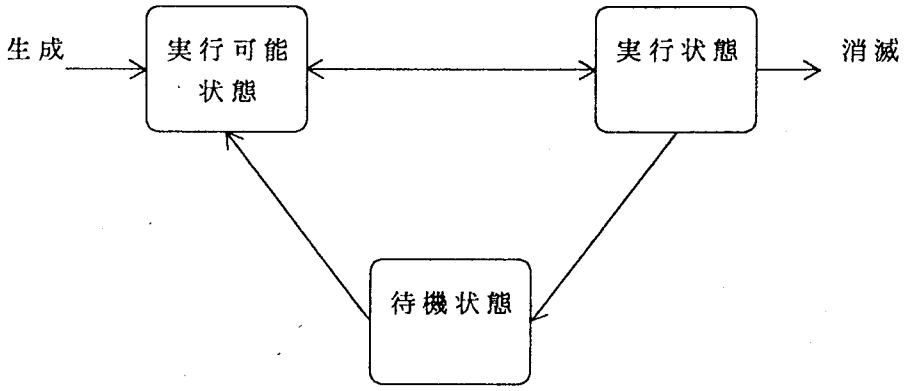
例えば、あるタスクの流れが、端末を介して利用者からの指示を待ち、指示が入ったら該当処理を行う。この処理にはデータの読み込みと画面への書き出しがあるものとする。

○プロンプトメッセージの画面への出力	実行→待機→実行可能→実行
○利用者の指示待ち	実行→待機→実行可能→実行
○指示された作業の実行	実行→
－ データ入力	待機→実行
－ 画面への表示	待機

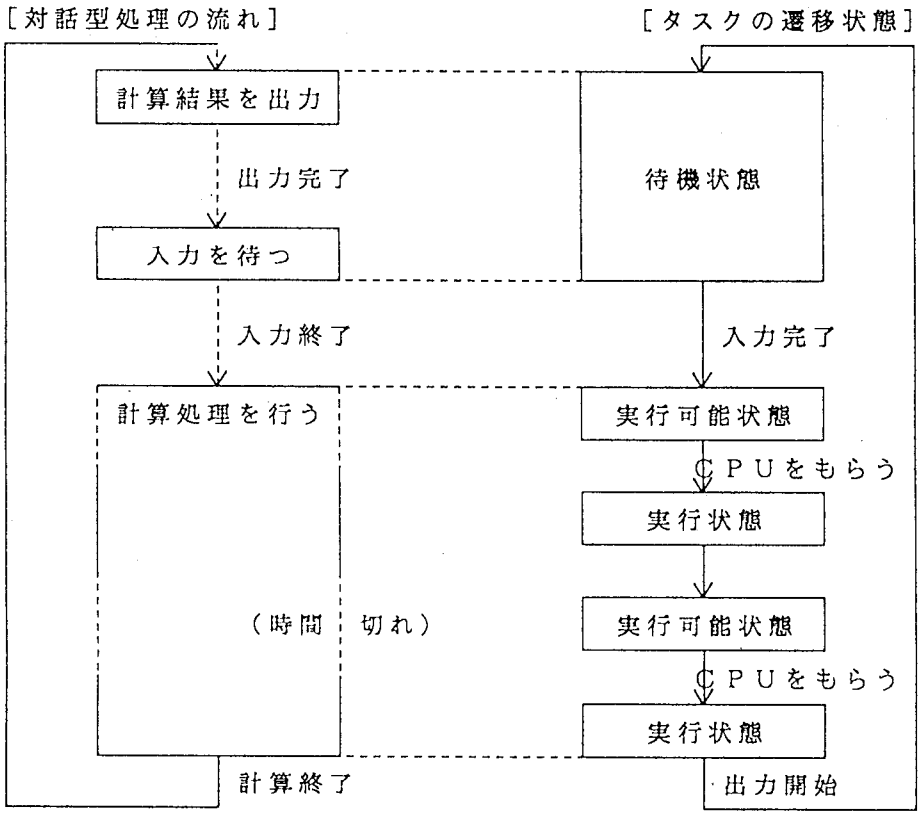
タスクの待機状態は、CPU を使う仕事を中断しなければならない状況を示している。タスクの状態遷移は、オペレーティングシステムによって生成されたあと実行可能な状態にはいる。この段階で、スケジューラが全ての実行可能な状態にあるタスクの中からその時点で最も優先度の高いタスクを実行状態に移行する。

実行状態のタスクは、入出力の時などには自ら待機状態にはいるり、入出力が完了すれば実行可能な状態に移され、次の機会に再び実行状態になる。また、実行中に時分割のタイムスライス値をオーバしたような場合には、オペレーティングシステムによって中断させられ、実行可能状態にもどされる。

図表Ⅰ－19　タスクの状態遷移



図表Ⅰ－20　対話型処理の流れとタスクの状態遷移の対応

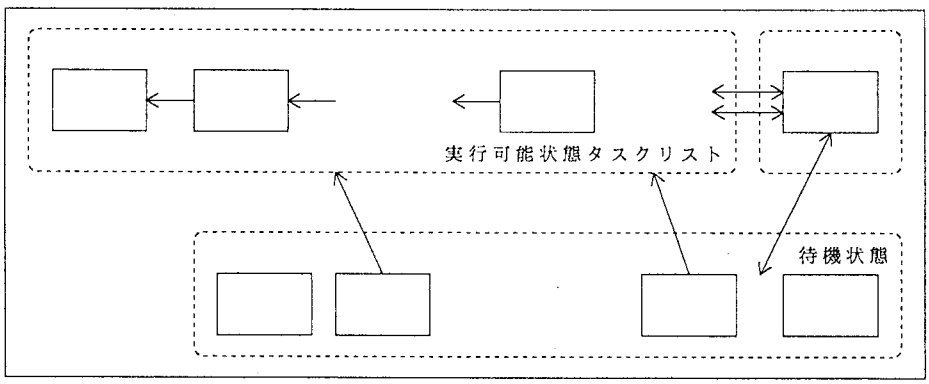


多重プログラミング下では、実行可能な状態のタスクや待機中のタスクは同一時点で複数個存在する。オペレーティングシステムは、それぞれのタスクをリスト構造で管理を行っている。これをそれぞれ実行可能タスクリスト（若しくは）キュー、待機タスクリストという。

タスクの遷移状態が変わるときは、これらのリストからの取り出し（デキュー）、若しくはリストへのリンク（キュー）操作が必要となる。どのタスクを実行可能タスクリストからデキューして実行可能にするかの判断はスケジューラが行う。これは、オペレーティングシステムの重要な機能の一つであり、どのようにスケジューリングを行えば最もコンピュータ資源の使用効率が高く、かつ、各タスクの所望する実行優先度に叶うかを判定している。

待機タスクリストからは、待機している事象の完了が通知されたときに、ただ一つのタスクだけが実行可能状態に移行するわけではない。同じ事象の完了を複数個のタスクが待機していることがある。

図表Ⅰ－21　タスクのリスト



（5） 対話型オペレーティングシステムの構造

典型的な対話型のオペレーティングシステムとしてUNIXを取りあげる。典型的なとは、世の中によく普及しているということであり、オペレーティングシステムの実現構造として典型というわけではない。MacintoshでもWindowsでも対話型オペレーティングシステムの典型ではあるが、内部はだいぶ異なっている。

まず、UNIX動作のアウトラインを見てみよう。UNIXマシンを立ち上げると通常ディスプレイにはユーザからのログインを促すプロンプトが出力される。ただし、ミニコンピュータなどのUNIXでは端末から最初のアテンションが入力されて始めてログインプロンプト状態となる場合もある。

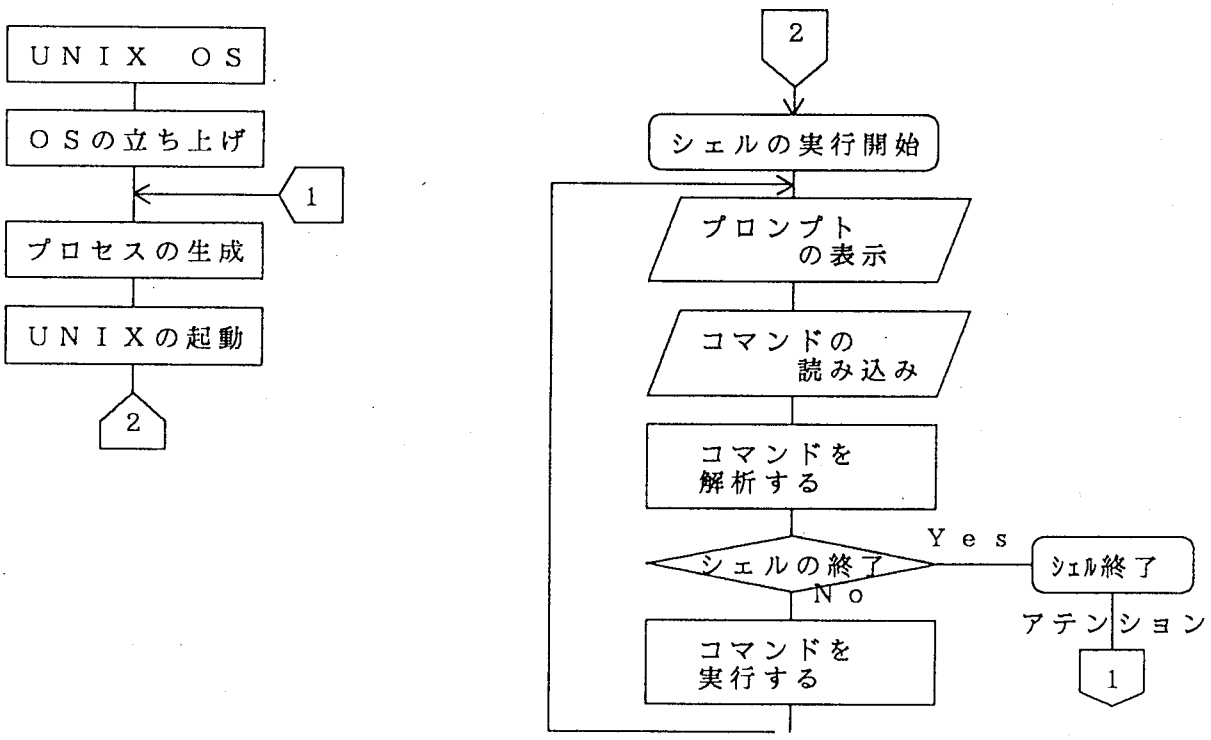
実は、このログイン促進状態となるまでは、UNIXOSは立ち上げに当たって色々な処理を行っている。OSの中核モジュールを主メモリにローディングする、ファイルシステムに矛盾がないかどうか確認する、各端末毎にプロセスを起こしてログインプロンプトを送信するなど。

利用者は、このプロンプトに対してコマンドを入力することとなる。プロンプトの標準文字はシステムのジェネレーション時に決められるが、利用者は自分で好みの文字に変更することができる。利用者がコマンドを入力するとUNIXのコマンドアナライザ（あるいはコマンドインタプリータともいう）がコマンドの文法や求められていることを解析して必要なプログラムを実行させる。UNIXではこれをシェル（Shell）と呼んでいる。

シェルは、UNIX立ち上げ後にOSによって最初に作られるプロセスであり、利用者のコマンド解析とその実行を司っている。図表Ⅰ－22に見られるようにUNIXシェルの動作は、プロンプト出力 → コマンドの入力と解析 → コマンドの実行 を一サイクルとしてログアウトするまで継続される。

コマンドは、ロードモジュールを読み込んで実行する場合と、頻繁に使われるようなものは主記憶装置に常駐していて直接実行される場合とがある。

図表Ⅰ－22　UNIX シェルの動作



起動されるコマンドは、シェルの子プロセスとして生成された後、対応するロードモジュールが読み込まれて実行に移る。シェルは、コマンドの終了を待っており、コマンドがエグジットするとシェルの待ちが解かれ、コマンドプロセスは消滅し、シェルは再びプロンプトとコマンド入力のフェーズにはいる。

UNIXは、プロセスが別のプロセスを生成できる。生成したプロセスを親プロセス、生成されたプロセスを子プロセスと呼ぶ。シェルは、どのようなプロセスに対しても生成元を遡れば先祖プロセスである。別のプロセスを作り出せる機能は、ほとんどの商用のオペレーティングシステムで用意されている機能である。

UNIXシェルでは一つのコマンド行に複数個のコマンドを並べることができる。コマンドの区切りにはセミコロン（；）を用いる。コマンド１；コマンド２；・・・と入力すれば、左から順にシェルが子プロセスを生成してはコマンドを実行することを連続的に行う。また、いくつかのコマンド行をひとまとめにしてファイル化し、コマンド入力先をそのファイルとして指定すれば、シェルはそれぞれ解釈して実行を制御することができる。これをシェルスクリプトもしくはシェルプロシージャなどという。これを使うと、種々のプログラムを複雑な手順で実行することが可能でもあることから、シェル言語などと呼ばれることもある。

UNIXでは、２個以上のプロセスを並列的に動かすことができる。方法としては、シェルコマンドで明示的に指定する場合と、プログラム内部から二つ以上の子プロセスを生成して同時動作させる方法である。

（6） タイムシェアリングシステム

バッチ処理から始まったコンピュータ利用形態に大きな変革をもたらしたシステムであり、プログラミングの全過程を端末を介して自らコンピュータと対話することにより、コンピュータへのジョブの投入および計算結果の受取に要するオーバヘッドをなくすことに貢献するものであった。

最初は、プログラミングおよびJCLのセットアップ、さらにはプログラムの実行にかかわり人での介在を最小化する点に利益をもたらしたが、

- プログラミング支援：テキストエディタ → 文法チェッカ、清書、静的解析
- 実行支援：デバッグライト → デバッガ

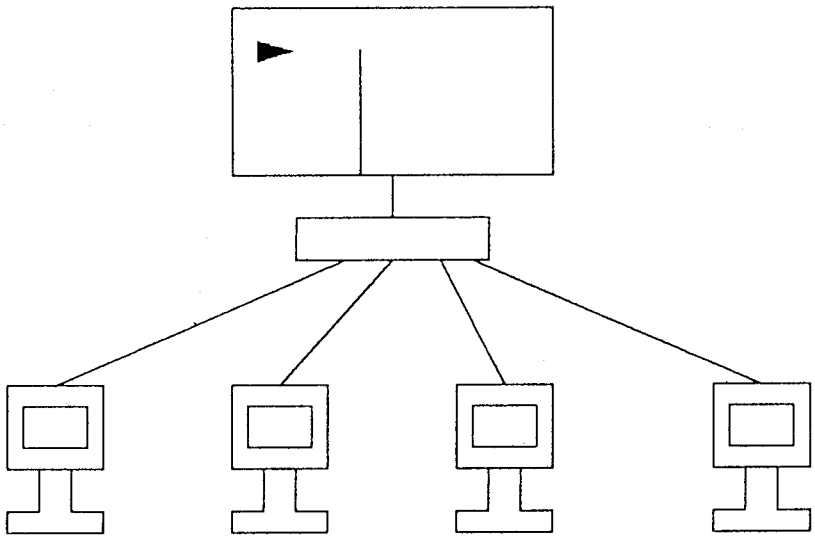
など、プログラミング生産性向上ツールの提供などに機能拡張され、対話プログラミングの重要性をますます高めることとなった。

タイムシェアリングとは、文字どおり計算機を時分割に用い、あたかも個々の利用者が同時にコンピュータを独占するような仕掛けを意味している。これは、OSの多重プログラミング機能を用いて実現されている。すなわち、各TSS端末に対して仮想資源を割り振り、瞬間的な一時点でコンピュータ独占状況を作り出している。

厳密にタイムシェアリングという言葉を定義するならば、バッチ処理であっても、リアルタイム処理であっても時分割機能によって多重性を提供していることに変わりはない。しかし、一般にタイムシェアリングとは、コンピュータの利用者が自分の体でコンピュータを同一時点で共同使用していることを実感できる場合を指している。

タイムシェアリングシステムの例としては、IBMのTSO、XEROX社のUTS、MITのMULTICSなどが商用化されたものとして実績を上げてきた。TSOは、IBMがバッチ処理技術に優れていたこともあり、比較的バッチ処理の延長線上で実現されたため、最初はプログラム生産性向上に対する高機能なサポートは少なかった。MULTICSでは、コンピュータの対話利用はどうあるべきかから十分な検討をされて作られたOSでもあり、今日のUNIX等の設計に多いな影響を与えてきた優れたOSであった。

図表Ⅰ－23　タイムシェアリングシステム



3．ユーザインタフェース

（1）　ユーザインタフェースの意義

汎用コンピュータと異なり、パソコンやワークステーションは基本的に個人々人もしくは数人のグループの持ち物である。価格が極めて安く、企業や社会生活における日常の業務処理をスムーズに遂行できるようなツールであることが永遠に求められる。

10年程以前のコンピュータでは、コンピュータへの指令や指示の形式は覚えるのに時間のかかるコマンドを使う必要があったし、また、計算処理結果も特別なコンピュータ用紙にプリント出力して紙を何枚も見ながら確認するといった手間のかかる作業であった。

近年は、ユーザインタフェース技術がどんどん改良され、覚え易くまた使いやすい操作で手間のかかる手作業を短時間にできるようになってきた。すなわち、操作のための学習時間が非常に少なくて済む、またしばらく使っていなくてもすぐに操作方法を思い出すなどの人間の生産性向上に大きく寄与していることになる。

ハードウェア面でみれば、キャラクタディスプレイからビットマップディスプレイになり、図形、グラフ、画像表現はもとより、マルチウィンドウ技術により一度に多くの異なる情報を確認することが可能となっている。これにはマウスの機能が大きく寄与している。

マウスのポインティング機能は、文字、図形、絵、グラフ、イメージなどの編集をいとも簡単にやってのける。最近では3次元マウスなどが出てきており、縦、横、高さ3軸方向でのポインティングが可能となり、さらには6軸マウスなども現れ上下、前後、左右回転など立体的な操作も可能となっている。

ペンコンピュータなども数年前から登場しており、文字入力の簡易化、自由メモの作成、各種書き物の極めて自由度の高い編集機能をも可能としている。

こういう中で、ユーザインタフェース設計の洗練さも求められており、J.S.Dumasは、良いインタフェース設計と悪いインタフェースについて以下のように指摘をしている。

－ ユーザインタフェース設計の7原則 －

［1］ ユーザを統制された状態におく。

［2］ ユーザの技能と経験のレベルを見定める。

［3］ 一貫性を保たせる。

［4］ ハードウェアやソフトウェアの内部処理をユーザにみせない。
［5］ オンラインドキュメントを提供する。
［6］ ユーザの記憶の負担を最小限にとどめる。
［7］ 優れたグラフィックデザインの原則に従う。

悪いインタフェースは、下記のようなものである。

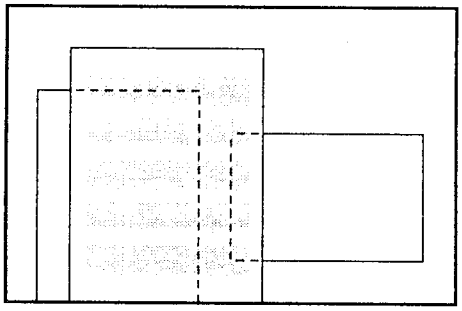
［1］ コンピュータが何を伝えようとしているのかよく理解できない。
［2］ アプリケーションの中での現在の位置づけがわかりにくい。
［3］ ユーザの指示に対してコンピュータがどう応答するかが明確でない。
［4］ 入力操作が複雑
［5］ 誤って操作したとき、元の状態に戻れない。

（2） 典型的なウィンドウシステム

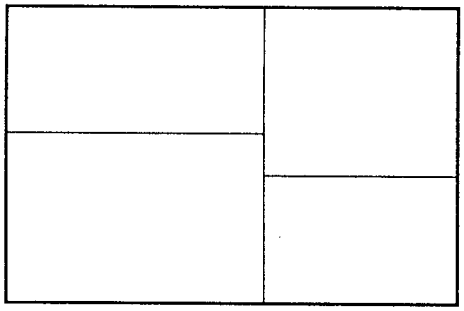
ウィンドウシステムとは、論理的にまとまりのある情報を一つの“窓”からディスプレー上に見せる技術である。しかも同時に複数個表示が可能である。ただし、ディスプレー上で一度に一種類の情報しか見ることができない場合、それを単一ウィンドウと呼ぶこともできようが、通常そういう表現はしない。

一度にいくつかのウィンドウが一つのディスプレー上で見ることができ、またそれぞれに対してあたかも同時に対話することができる仕組みを持っている。これには、ディスプレーが物理的に完全に分割されウィンドウの重なりがない“タイリング方式”と、ウィンドウが重なる“オーバラッピング方式”とがある。

図表Ⅰ－24 ウィンドウシステムの二つの方式



a．オーバラッピング方式



b．タイリング方式

タイリング方式は、初期の（あるいは原始的な）ウィンドウシステムであり、基本的にはウィンドウが重なるという実現方式ではない。この方式は、キャラクタディスプレイ時代の考え方であるが、

・ コンピュータ利用の一つの仕事で入出力待ちのような状態となり、その時点で利用者がコンピュータに触れないというのはいかにも不便であることを解消した、
・ 複数個の仕事を同時にできる、

ことは大きな進歩であった。

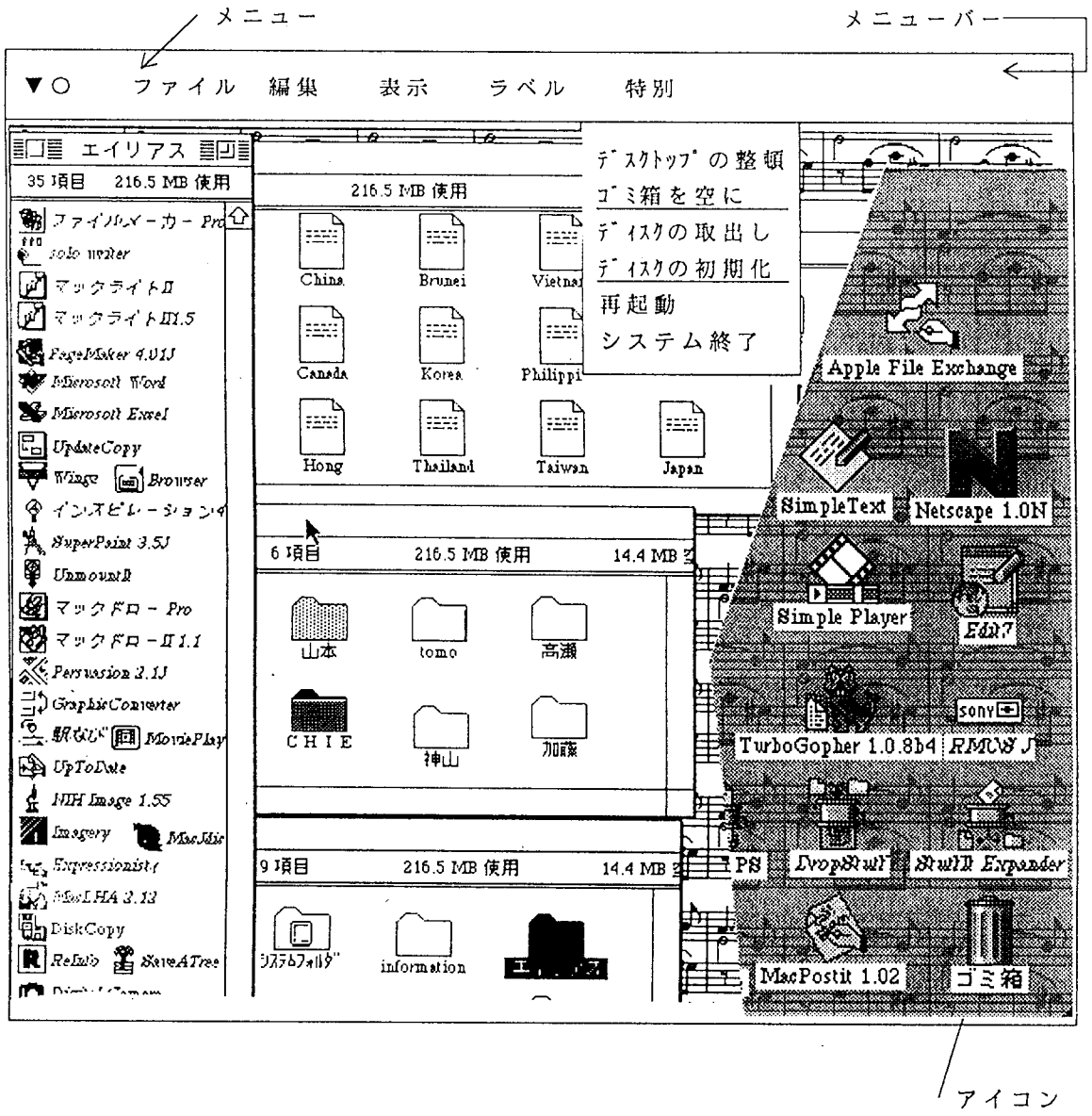
オーバラッピング方式は、ビットマップディスプレーの出現によりコンピュータの操作性を格段に向上させた。ウィンドウ内にどのような大きさのドキュメントを割り当てることができ、ウィンドウの重ね合わせも可能で、しかも文字（テキスト）、図表、図形、画像などが同一のウィンドウで扱える点はワークステーションやパソコン利用に大きな革新を与えた。

WindowsやMacintoshあるいはOS／2はマルチウィンドウシステムを提供する典型的なオペ

レーティングシステムである。ただし、マルチウィンドウであるからといっても、どのオペレーティングシステムにおいても同じ使いがってというわけではない。ウィンドウ上での表現方法は、知的所有権なども主張する対象ともなっており、各社それぞれちょっとずつ異なる実現方法をとっている。

以下でMacintoshについてウィンドウシステムの例を見てみよう。

図表Ⅰ－25　ウィンドウの例



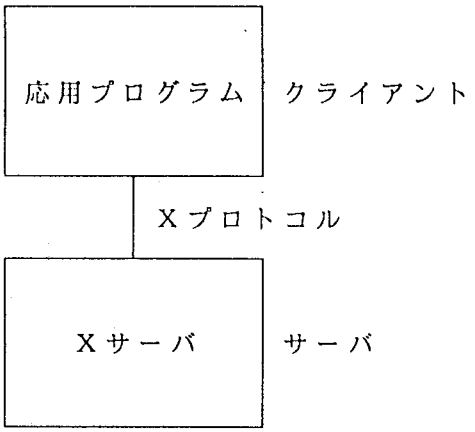
（3）　ユーザインタフェースシステムの例

マウスを操作し、アイコンを用いて文字列や図形の各種編集を行ったり、視覚的に情報を表示したり、判断したりしながら各種のプログラムを動かしたりするための操作法をグラフィカルインタフェース（GUI）という。

標準的なグラフィカルユーザインタフェースの代表的なシステムとしてOSF／Motifがあげられる。これは、Xウィンドウ上に高いレベルのGUIを提供するものであり、Motifウィジェットと呼ばれる。製品としては、ポップアップメニュー、プッシュボタンなどを作成するためのライブラリを用意している。

Ｘウィンドウシステムは、クライアントサーバモデルに基づいて設計されており、ネットワーク上でハードウェアベンダやソフトウェアベンダの製品依存を超えて透過性が実現されている。

図表Ｉ－26　Ｘウィンドウシステム
クライアントとサーバ

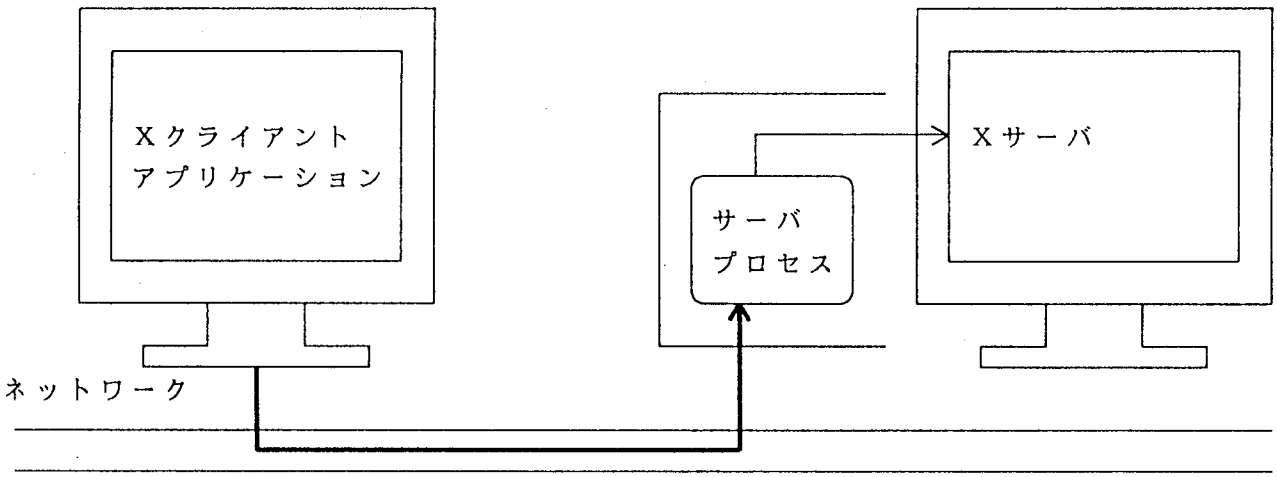


Ｘウィンドウシステムでは、ウィンドウ機能を提供するものをＸサーバと呼び、それを利用するすべてのアプリケーションプログラムをＸクライアントと呼ぶ。

Ｘクライアントは、Ｘサーバに対して、Ｘプロトコルを用いて逐次要求を出し、ウィンドウの表示制御を行う。

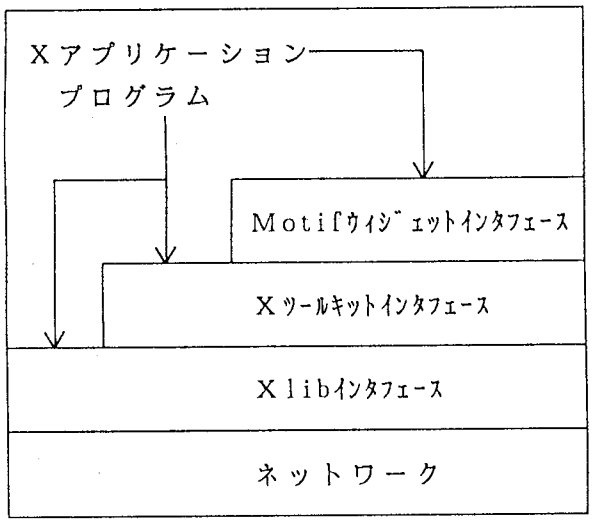
Ｘサーバがすべてウィンドウ制御を行うため、Ｘクライアントアプリケーションはディスプレイやキーボードの制御を直接行わず、必ずＸサーバに依頼して代行してもらう。

Ｘクライアントアプリケーションはウィンドウ制御の依存性がなくなるため、繁雑さ、マシン依存性から独立化され、容易にかつ移植性の高いプログラムを作ることができる。



Motif、Ｘウィンドウのアプリケーションは、図表Ｉ－27のようなインタフェースでプログラム開発が行われる。Ｘウィンドウシステムは、ネットワーク対応となっており、ベースにネットワークがあり、その上にXlibというプリミティブな関数群がある。さらに、その上にはマクロ的な機能としてＸツールキットが提供されており、それを使うGUIとしてMotifウィジェットが提供される。

図表Ⅰ－27　応用プログラムとXウィンドウシステムのインタフェース



Xアプリケーションプログラムは、ウィンドウにボタンを提示し、そのボタンがマウスクリック選択されたときに、対応する操作を行う。

Xlibは、Xウィンドウシステムにアクセスするためのプリミティブなライブラリであり、その呼出手続きは標準化されている。Xウィンドウの全ての制御が可能である。

Xtoolkitは、あるまとまったXlibコールシーケンスをイントリンシックとして提供する。

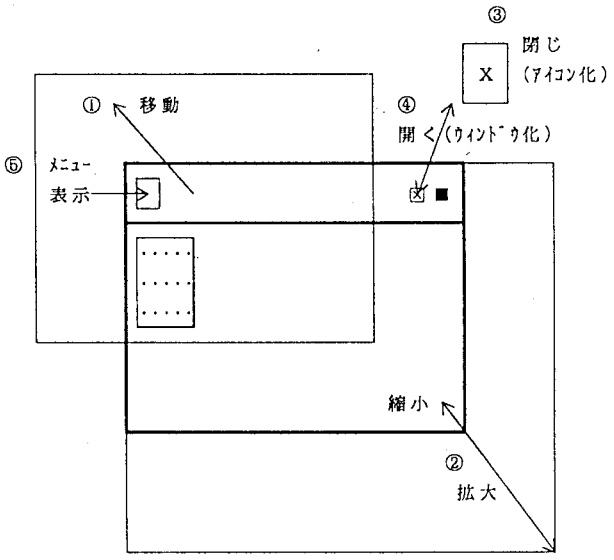
Motifウィジェットは、Xtoolkitイントリンシックによって実現するユーザインタフェースのセットである。

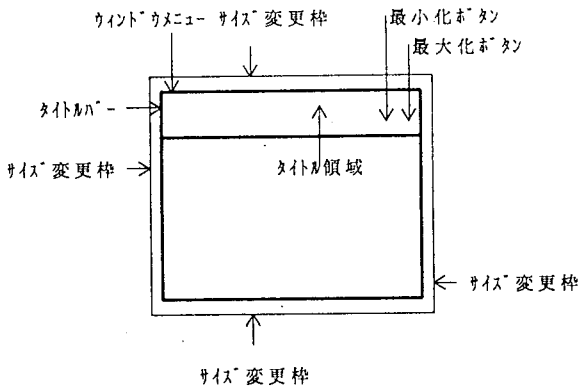
以下で現在注目されている代表的なGUIについて説明する。

［1］ UNIX

UNIXマシンでは、OSF／MotifとOPEN LOOK GUIが代表的なGUIであった。しかし、結局、商用としてはOSF／Motifが生き残った。

図表Ⅰ－28　ウィンドウの操作例

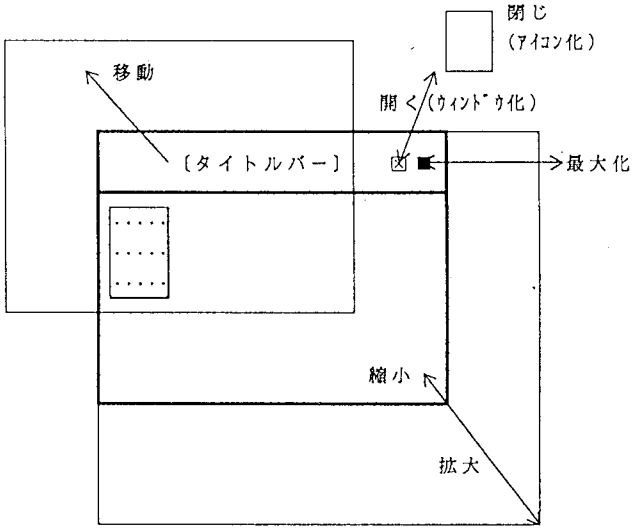




① ウィンドウの移動
マウスカーソルをウィンドウのタイトルバー上に置いてドラッグ
② ウィンドウサイズ変更
マウスカーソルをウィンドウのサイズ変更枠に置いてドラッグ
③ アイコン化
最小化ボタンをクリック
④ アイコンのウィンドウ化
ウィンドウ化対象のアイコン上にカーソルを置き、ダブルクリックする。
⑤ ウィンドウメニューの表示
ウィンドウメニューボタンをクリックするとメニューがポップアップされる。

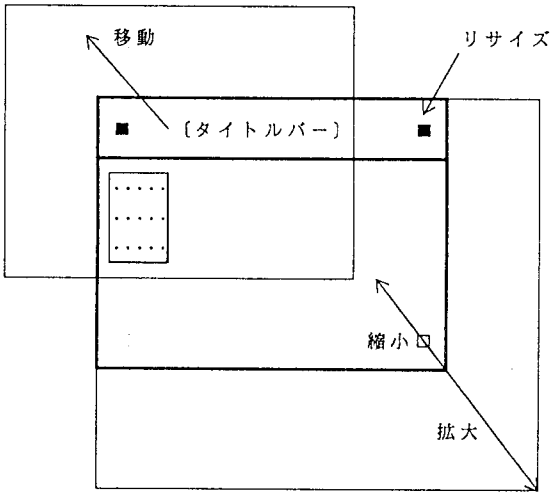
［2］ Windows

図表Ⅰ－29 ウィンドウの操作例



［3］ Macintosh

図表Ⅰ－30 ウィンドウの操作例



4．主要用語

イベント　事象　イベント駆動
多重プログラミング　マルチプログラミング　単一プログラミング
システム資源　システム資源管理機能　時分割
タイムスライス　見せかけコンピュータ　仮想記憶
プロセス　タスク　ジョブ
実行状態　実行可能状態　待機状態
入出力管理機能　状態遷移図　リスト
スケジューラ　コマンド　プロンプト
シェル　ログイン　ログアウト
セッション　親プロセス　子プロセス
シェルスクリプト　シェル言語　タイムシェアリングシステム
TSS　時分割システム　ビットマップディスプレイ
キャラクタディスプレイ　ウィンドウシステム　オーバラッピング方式
タイリング方式　タイトルバー　メニューバー
メニュー　プルダウンメニュー　ポップアップメニュー
マウスカーソル　クリック　ダブルクリック
ドラッグ　グラフィカルユーザインタフェース
GUI　ルックアンドフィール　Xウィンドウシステム
クライアントサーバモデル　サーバ　クライアント
ウィジェット