TECHNOLOGY REVIEW

UNISYS 技報

通巻 49

1996年5月発刊 Vol. 16 No. 1

特集：オープン・エンタープライズ・サーバ
——基本思想と要素技術——

巻頭言

特集「オープン・エンタープライズ・サーバ
——基本思想と要素技術」の発刊によせて ……堀川二三夫 1

講　演

21世紀を目指す企業情報システム ……C. Morris 3

論　文

新世代メインフレームの基本思想 ……今江　泰，丑久保年常 19

シリーズ2200における
大規模トランザクション処理を実現する要素技術 ……城川孝二 37

Aシリーズにおける
大規模トランザクション処理を実現する要素技術 ……森　良行 56

大容量データ処理を実現する入出力機構 ……白崎　宏，新津昭義
坂本徳明，金井秀量 71

並列システムにおける負荷平準化機能 ……畑　邦明 99

オープン・プログラミング環境(OPE)
——OS 2200との共存の仕組みと意義 ……元山裕之 112

Aシリーズの次世代オープン・
エンタープライズ・サーバの実装技術 ……室木　通，原　広仁 124

シリーズ2200オペレーティング・システムの発展 ……伊藤龍彦 139

Aシリーズオペレーティング・システムの発展 ……鈴木秀明 162

解　説

ITASCA 3800シリーズのハードウェア技術 …… 183

討論会

エンタープライズ・サーバの現状と展望 ……遠藤和弥，馬場功二
神谷是公，今江　泰
田辺英夫，松倉　司 196

新製品紹介 ……213

掲載論文梗概 ……表2，3

日本ユニシス

ISSN 0914-9996

21世紀を目指す企業情報システムは平成7年9月14日に開催された「エンタプライズ・サーバ・フォーラム95」においてガートナー・グループのC. モリス氏によって行われた講演録である. 本講演では，システムの分散化，集中化，C/Sシステムが今後どうなるか，それに対して情報システム部門はいかに対応すべきかを示している.

今江泰・丑久保年常の**新世代メインフレームの基本思想**は，今後の企業情報システムの形態を新世代メインフレームとオープン・システムとがそれぞれの特徴を生かしつつ共存・連携するエンタープライズ・クライアント/サーバ・システムととらえ，そこでのエンタープライズ・サーバとしての新世代メインフレームの位置付け，有効性，求められる要件，要件を実現する要素技術について述べている.

オペレーティング・システムの役割は，適切な仮想化を通してシステム資源を抽象概念として使用者に示すとともに各資源を最大限に利用することにある. 城川孝二は**シリーズ2200における大規模トランザクション処理を実現する要素技術**の中で，シリーズ2200のOSであるEXECが採用している様々な技術の一端を，トランザクション処理に焦点を当て，構造解析レベルではなく，一般論やUNIX[*1]との対比等も交えて，紹介している.

森良行の**Aシリーズにおける大規模トランザクション処理を実現する要素技術**は，AシリーズのOSであるMCPが提供している多重プログラミング環境を提示し，その環境下でトランザクション処理システムを支える基盤ソフトウェアが行っている多重化処理の仕組みについて述べている.

メインフレームを中核としたシステムは入出力分野が特に強いと言われている. 白崎宏・新津昭義・坂本徳明・金井秀量の**大容量データ処理を実現する入出力機構は**，トランザクション処理に於ける入出力の応答性能・多重度の観点から入出力性能のシステム全体の性能に対する重要性を考察し，先進的な入出力機構（シリーズ2200のXPC及びAシリーズの機能別入出力プロセッサのアーキテクチャ），メインフレームOSが採用している入出力技術について述べるとともにメインフレームが支援する各種チャネルを紹介している.

メインフレームによる並列処理システムでは，その性能を十分に引き出すためには，クラスタ間の負荷を平準化制御することが必要である. 並列トランザクション処理システムではCPU利用率を，バッチの並列処理システムではCPU利用率・I/O利用率・メモリ利用率を平準化制御する必要がある. 畑邦明の**並列システムにおける負荷平準化機能は**，平準化制御を行ういくつかの方法を解説し，次に並列トランザクション処理システムで負荷制御を行うバランスマネージャの仕組みについて，最後にバッチの並列処理とその平準化制御について考察している.

シリーズ2200上のUNIX OSであるOPEは，OS 2200と共存することにより，OS 2200上のユーザの資産を継承しつつ，これらの資産とオープンな世界との連携を可能にしている. 元山裕之は**オープン・プログラミング環境(OPE)——OS2200との共存の仕組みと意義の**中で，OPEの機能概要を紹介するとともに，OS 2200との共存の仕組みや意義を明らかにしている.

AシリーズではTCP/IP関連製品，OpenOLTP，ODBCなどオープン製品やCCE/CCPと呼ばれる独自の協調コンピューティング環境を提供することで，Windows[*2]及びUNIXとの協調処理環境に積極的に対応してきた. とくに昨今のMicrosoft製品の影響は大きく，次世代オープン・エンタープライズ・サーバはこのような市場を背景に生まれた. 室木通・原広仁の**Aシリーズの次世代オープン・エンタープライズ・サーバの実装技術**は，このサーバの目指す所，ハードウェア/ソフトウェアでの実装技術，将来のメインフレーム像に触れている.

特集「オープン・エンタープライズ・サーバ ——基本思想と要素技術」の発刊によせて

堀　川　二三夫

これまでの数十年間，情報処理システムの構築に際してメインフレームが大きな役割を担ってきた事は確かな事実であるといえよう．(ここでは，メインフレームの定義を提供者独自のハードウェア・アーキテクチャとオペレーティング・システムにより構成されているコンピュータ・システムと考えている．) しかしながら，最近の情報処理システム構築におけるダウンサイジング要求，オープン化要求の動きは大きなものがあり，メインフレームとしてもこれらの要求に対応すべくそのあり方を自ら変えつつある．すなわちメインフレームからエンタープライズ・サーバへ，さらにオープン環境における適合性を強化したオープン・エンタープライズ・サーバへの進展である．

このような状況認識に基づき，次のような事項について当社の技術の観点からの取りまとめを行うべく，本特集号を編集することとした．

1) オープン・エンタープライズ・サーバを巡る環境はどのようになるか，また，今後のあり方をどのように考えるべきかを整理する．
2) オープン市場のプロダクトと比較してオープン・エンタープライズ・サーバの方が優位性を保持している要素技術部分を整理する．
3) オープン・エンタープライズ・サーバを支えてきたオペレーティング・システムを技術史的観点から整理する．

今後ますます多様化するであろうコンピューティング環境の中で，どのプラットフォームを採用するのが最適かを検討する局面がしばしばあるものと考えられる．このような場合における参考情報の一つとして，オープン・エンタープライズ・サーバの特質を理解していただくために上記の観点からとりまとめを行おうとしたものである．

オープン・エンタープライズ・サーバを支えるオペレーティング・システムは，日本における顧客の基幹業務を支える機能を提供すべく，30年にわたる進化の過程を経てきている．この点においてオープンのプロダクトに比較してオープン・エンタープライズ・サーバの優位性が現時点では未だ存在すると言える．具体的にどのような部分に技術的優位性があるかを明らかにし，どのように構成されているかを提示することは，早い時期にオープンのプロダクトが具備すべき機能が何であるかを示すことになる．別の表現をするならば，オープンのプロダクトがこれらの機能を実現しない限り，従来オープン・エンタープライズ・サーバが担ってきた基幹系業務を円滑に受け取ることが難しい部分が出てくると言うことである．

オープン・エンタープライズ・サーバを巡る環境については，当社の調査・分析報告を掲載するよりは，中立・公正な第3者の客観的報告に委ねるのが適切であろうと考え，当該分野で

の著名なコンサルタント会社であるガートナグループの講演録を掲載することにした．また，今後の方向性については，オープン・エンタープライズ・サーバが必要とされる要件について明らかにする中で理解していただくものとし，あえて「新世代メインフレームの基本思想」と題した論文によることとした．

オープン・エンタープライズ・サーバが優位性を保持していると考えられる要素技術については，いわゆる基幹系業務を実行するのに必須である二つの機能について言及した．その一つは応答時間の保証が要求される大規模トランザクション処理を実現する技術についてであり，もう一つは大容量のデータを管理する技術についてである．「大規模トランザクション処理を実現する要素技術」「大容量データ処理を実現する入出力機構」がこれにあたる．

オープン・エンタープライズ・サーバをオープン環境にさらに適合させることを意図して提供される機能については二種の論文を準備した．さらに，今後の並列処理コンピューティング環境に対応するための機能を紹介する論文も準備した．また，これらのソフトウェアを搭載するハードウェアとしてのITASCA 3800がどのように実装されるかを示す論文をも用意した．これらの論文を一読していただくことにより，当社が提供するオープン・エンタープライズ・サーバがどのような進展を意図しているかの一端を見ていただきたいと思っている．

オペレーティング・システムの発展の経緯を俯瞰するものとしてシリーズ2200とAシリーズそれぞれについての歴史を取りまとめた．これはメインフレームがたどった道筋であり，今後オープンのプロダクトが進化していく際の指針となりうるものである．

本特集号では，従来にはなかった「討論会」を掲載している．オープン・プロダクトとオープン・エンタープライズ・サーバとの相違点や今後のコンピューティング環境の中でそれぞれがどのような役割を果たしていくべきかを参加者が自由に語り合ったものである．

このような構成で本特集号を編集したが，今後のオープン・エンタープライズ・サーバのあり方をご理解いただくのに十分であるとは言えないかとも思っている．皆様のご批判をいただきたい．

(システムプロダクト部 部長)

21世紀を目指す企業情報システム

Information Systems Divisions Toward 21st Century

Chris Morris

要　約　本稿は，日本ユニシスにおいて平成7年9月14日に開催された「エンタープライズ・サーバ・フォーラム'95」でガートナ・グループのクリス・モリス氏によって行われた講演「21世紀を目指す企業情報システム」の講演録である．

本講演はシステムの分散化，集中化，C/SSが今後どうなるのか，それに対して情報システム部門はいかに対応すべきかを示している．

1) 近年LANを使った分散コンピューティングが進んでいるがメインフレームほど可用性は上がらない．そのため現在，再度メインフレームによる集中化の方向へ進む動きが出てきた．ただし，C/SSは価値のあるシステムなので，C/SSが完全に集中化に戻るということはない．

2) 大企業では最近分散型サーバをデータセンタに置き換え始めている．それは，データセンタに設置した方がセキュリティ保護が万全であり，バックアップ体制も充実しているからである．

3) C/SS化は従来の集中型システムより低コストで実現できると言われていたが，調査結果では，メインフレームでやるよりコストがかかる実態が明らかになった．

4) 今後5年間は，継続的な変化・複雑化・混沌という三つの言葉がキーワードになる．エンドユーザ部門は新しい技術を理解し導入することが次第に難しくなっていく．これから情報システム部門にとって大切なのはエンドユーザに対するサポートおよびエンドユーザに対して明確な方向付けを示すということである．

Abstract This article is proceedings of "Information system divisions toward 21st century" which was presented by Chris Morris who belongs to Gartner Group at "The enterprise Server Forum '95" held on September 14, 1995. He indicated the future direction about distributed systems, centralized systems and C/S systems. And furthermore, he shows the way how to cope with these changes as the role of information system divisions.

1) Distributed computing systems using LANs are increasing recently. However their availability is less than the mainframe systems. Therefore the movement which promote to back to centralized systems using mainframes is underway now. On the other hand, C/S systems do not fully back to centralization systems because the C/S systems are very valuable systems.

2) Recently big business are replacing distributed servers with enterprise servers at data centers. Because the enterprise servers at data centers have two advantages.

One is the security point of view and the other is backup operations.

3) Though it is said that C/S systems can be created more cheaper than the centralized systems, one survey shows the opposite result. That is, mainframes are less expensive than C/S systems.

4) Keywords in five year period from now on will be "Continuous Change", "Complication" and "Chaos". End user divisions will have difficulty to understand and install new technologies gradually. From now, important rolls of information system divisions will be to support end user divisions

and to indicate the clear direction.

■情報技術の進展とビジネス環境の変化

ガートナ・グループの顧客の90%近くが，情報技術（IT）に関わっている企業なので，日本の情報システム部門（IS部門）の皆さんの前で話す機会を得たことを大変喜ばしく思っている．さて，この5年間，コンピュータ業界は非常に大きな変革を遂げてきた．この変化は，過去20年間経験した中でも一番大きな変化のうねりとなっている．それは情報技術だけでなく，ビジネス環境も巻き込んだ形で変化しているからである．そして情報技術がビジネス環境の変化をもたらした大きな要因となっている．

その結果，情報処理に携わっている人達は，エンドユーザへの対応がますます難しくなっている．エンドユーザからはもっと大きな機能が欲しいとか，アプリケーションの面でももっといろいろな処理がしたいといった要求が出ている．ところが“予算”の方は少しも伸びてくれない．逆に縮小されることもある．そうした中で，エンドユーザに対して，いままで以上にサービスを提供する必要が出てきている．

従来，メインフレーム・コンピュータは，性能や信頼性も高く，99.9％以上の可用性があった．そして，近年LANを使った分散コンピューティングが進んでいるが，メインフレームほど可用性は上がらず90～92％に止まっている．したがって，ミッション・クリティカルなアプリケーションをLANで提供しようとするとエンドユーザが満足する可用性に達せず，文句を言われる状況になっている．

そのようなことから現在，再度メインフレームによる集中化の方向へ進む動きが出てきた．ここ数年間は集中化されていたシステムを分散化する傾向にあったが，またそれを集中化に戻そうという動きである．

■C/SSが進めるオープン化とコモディティ化

クライアント/サーバ・システム（C/SS）は価値のあるシステムなので，C/SSが完全に集中化に戻るということはないが，C/SSには，さらにオープン化を進行させ，そしてコンピュータ技術のコモディティ化を進めるという傾向が見られる．

オープン・システムの定義について，われわれは，「アプリケーションの移植が可能であり，相互運用性があり，そして業界で事実上受け入れられている標準に則って作られたシステムである」と定義している．

約10年前にオープン・システムが普及し始めた頃は，UNIX*1が注目を集め，UNIX一辺倒であった．UNIXの方に目を向けないと，将来は死んだも同然だとよく言われた．

そして，各ベンダーはユーザからのオープン化ニーズによく対応し，業界スタンダードのAPIを導入した．1996年(来年)までに，このAPIを各ベンダーのOSに採用したものが提供されるようになる．つまり，XPG 4のSPEC 1170に準拠したOSが各ベンダーから提供される．それが来年には実現するということである．

将来，情報システム（IS）部門は，どのような形になってしまうのかという課題がある．つまり，オープン化が進み，OSのAPI化が進むと専用システムがなくなってしまい，IS部門の存在価値がなくなってしまうのではないかという恐れである．

しかし，昨年11月にC/SSを利用している人を対象に，「メインフレームの上のアプリケーションをどのくらいオフロードしたいか」という調査を行ったところ，70％以上の人がメインフムームのAPを使いたいと答えている．ところが実際には全体の4％しかオフロードした人はいない．C/SSの方に，アプリケーションを移したいという意思があってもその程度しか行われていないのが実態である．

C/SSは，通常考えられているものよりも非常に複雑な構成となり，大変コストがかかるものである．メインフレームと同じ規模のシステムで約3倍のコストを要するといわれている．

それとC/SS化は，CPU，ストレージ，そしてプラットフォーム等，情報技術をコモディティ化する．

たとえば，インテル社からチップを買ってきて，簡単に，安価な様々なプラットフォームを作ることが可能になっている．そして，コンピュータ市場には数多くのベンダーが参入し価格競争を引き起こしている．価格競争の結果，ハードウェアの価格は安くなったが，一方で，体力のない，小さなベンダーはマージンが少なくなり，倒産するという事態が見られるようになってきた．

ストレージの市場もコモディティ化がどんどん進んでいくと思う．PCでは3.5インチのディスク・ドライブが，データセンタではレイド・ディスクが使われているが，1999年までには1/4インチのディスク・ドライブが使われることになろう．

コモディティ化により，ディスクの開業・製造が容易に行えるようになった．その結果，MIPSとか，メガバイトのディスクをまるでタダみたいな値段で手に入れることができるようになるだろう．このように，ハードウェアの単価は，ますます下降傾向をたどっていこう．

■今後の変化

この業界が2000年までの今後5年間でどう変わっていくかは，①継続的な変化，②複雑化，③カオス一混沌という三つの言葉がキーワードになる．そしてエンドユーザ部門は，新しい技術を理解し導入することが次第に難しくなっていく．このためIS部門の人達はその対応に苦労するだろう．ガートナ・グループでは，C/SSアーキテクチャについて，この2年間注目してきた．このセミナーを聞きにきている企業の皆さん方でも40％がIT(情報技術)アーキテクチャのドキュメントを持っていれば良いほうだろう．

従来，各ベンダーからITのアーキテクチャを買うというのがこれまでの方法であった．しかもこの方法に自信を持っていた．新しいITを買えば，他の場面でも，それが通用していた．

しかし，今は違ってきている．現在は，各ベンダーからコンポーネントを買うようになった．それらのコンポーネントが必ずしも一緒に使えるかどうかの保証はない．したがって，自分達が購入したコンポーネントについて不安を覚えている．

■短命化する製品のライフサイクル

もう一つ重要なのは，プロダクトのライフサイクルである．1990年の調査では，メ

インフレームは7〜8年間は使えるだろうと言われていた．しかし今や3年持てば良い方である．デスクトップの機械は，2年も経つと陳腐化してしまう．これは環境の大きな変化と言える．だからIS部門としては，こうした変化の対応に毎日のように追われているものと思う．APも同様で，せっかく作ったAPが使えないという事態が起こっている．それだけビジネス・モデルの変化も激しいわけである．そして，変化というものは決して終わるものでなく，常に継続して変化している．その変化にAPを適応させていかなくてはならないという問題が生じているのである．

■トップダウン型とボトムアップ型の購買パターン

こうした新しい時代において，トップダウン型とボトムアップ型の二つの処理パターンの衝突が見られるようになった（図1参照）．

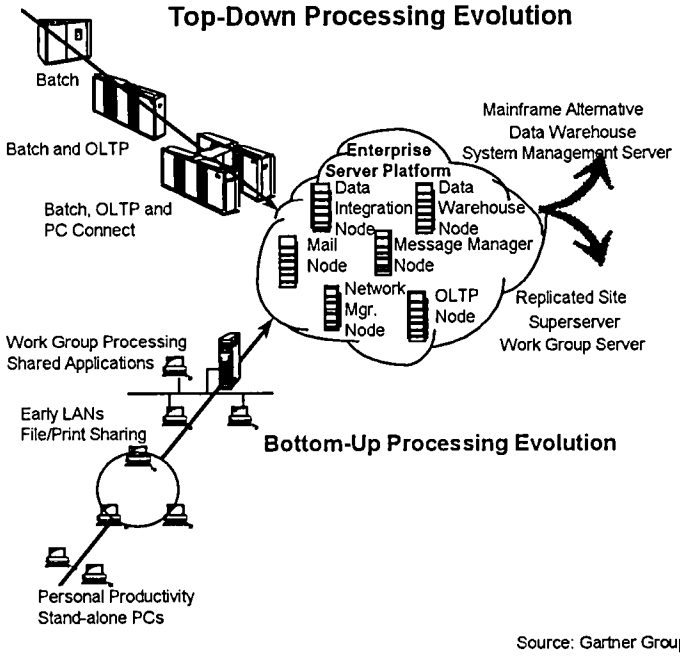


図1　トップダウン型とボトムアップ型の二つの処理パターンの衝突

伝統的な購買パターンは，中央のIS部門が企業全体にわたる機器の購入をコントロールしてきたやり方である．

ところが10〜12年前からPCが企業内に普及し始めた．その結果，エンドユーザ部門のマネジャーが自分たちの部門で使う機器やシステムを買うというボトムアップ型の購買パターンが出てきた．そして，この二つの購買パターンが混在するようになってきた．つまり，エンドユーザ部門が購入する機器やシステムのコストが高くなり，平均して年間50万ドルに及ぼうとしている．したがって中央のIS部門は，当然，これを何らかの形でコントロールしたいと考える．

最近の調査では情報技術関連の総予算のうち50％以上はエンドユーザがコントロールしているといわれている．それに対して，IS部門は，どういったアーキテクチャ

の下に，どういった機器をエンドユーザ部門が買ったらよいかといった，ブループリント一青写真を作り，指定する動きが出始めてきた．

■エンタープライズ・サーバの登場

このような状況の中にエンタープライズ・サーバというプラットフォームが登場してきた．

これは，大きなメガ・サーバということもあり，それぞれのユーザニーズを組み合わせて利用していくというケースが見られる．

その一つとして意思決定支援システム（DSS）のために用いられる超並列処理型のエンタープライズ・サーバが普及してきた．意思決定支援のためには膨大なデータをDB化し，それを迅速に処理する必要があるからである．

また，電子メールを独立した別のプラットフォームで走らせるというように，複数のサーバが使用される時代になってきた．複数のベンダーからいろいろな機器を買い，しかも，エンドユーザがプラットフォームを管理するケースも見られるようになってきた．

5〜6年前，分散処理方式が登場した頃，データを使う部署にデータを置くべきだとよく言われた．しかし，今やそれはあまり問題になっていない．ユーザは，どこにデータが存在するかということは気にしないからである．つまり，迅速に，信頼する形でデータをアクセスできれば，データはどこに存在していてもかまわないと考えるようになってきた．そこで，再びエンタープライズ・サーバをデータセンタで使おうという考え方に戻ってきたのである．

最近，特に大企業では分散型サーバをデータセンタに置き換え始めている．それはデータセンタに設置した方がセキュリティが確保できるからである．データセンタは，セキュリティ保護が万全だし，バックアップ体制も充実している．

■C/SSとデータ・バックアップの問題

私は本年3月にメルボルンにある大手銀行のIS部門のマネジャーに呼ばれて話をしたことがある．彼はオーストラリア国内をはじめニュージーランド，イギリス，それからシカゴの支店を含め，銀行内のデータがどこに存在しているかを調査した人で，その結果，銀行データの実に55％が中央のデータセンタ以外の外部に存在していることを発見し，非常に心配になってきたと言うのである．たまたまエンドユーザの人に会い，話を聞いたところ，そのユーザは「3か月前から全くバックアップ・データをとってない」と言う．「どうしてか」と聞くと，「エンジニアの人がきて，ハードウェアやソフトウェアのアップグレードを行い，バックアップの自動化を行った．しかし，クリスマスと新年の休み明けに再開したとき，それがうまく動かなかったので，それ以後バックアップはとっていない」と言うのである．

銀行のデータというのは，ミッション・クリティカルなデータである．ところが，そのデータのバックアップがとれていないというのは驚くべきことである．したがって，中央のデータセンタがアプリケーションやデータをコントロールすべきであるという重要性が，この一例にも見られるわけである．

それと小さなサーバを数多く持つよりも，大きな機械やサーバをすべて統合化した方が経済的であり，管理をする上でも能率的である．

■情報化投資の半分は分散コンピューティングに

図2は，今後，情報技術にどのくらい投資するかを調査し，まとめたものである．

皆さんに「会社の総売上げのうち情報化関連にどのくらい投資しているか」と聞けば「多分1〜2％ぐらいでしょう」と答えるだろう．これは妥当な数字である．しかし，われわれの調査によると実際には売上の5〜6％は使っているのである．データセンタの予算や使った金額の追跡は容易にできるが，分散コンピュータ環境（図2のグラフの中央部分）になってしまうと，全く違った状況になってしまう．全情報技術投資のうちの3分の1が分散コンピューティングに使用されている．1998年になるとその割合はさらに高まり，総売上の4.6％を占めるようになると予測されている．全情報化投資のほぼ半分が分散コンピューティングに使われるということである．

グラフの一番下の部分は，情報化投資予算に入っていない支出である．例えば，ランチタイムのときに，どこかのPCショップに行き，何かのソフトウェアを買ってきて，それを経費として落とし経理処理された数字である．そうした隠れた資源への投資も10％増えると予測されている．

そして情報化投資全体で14％は増えていくだろうと見込まれている．

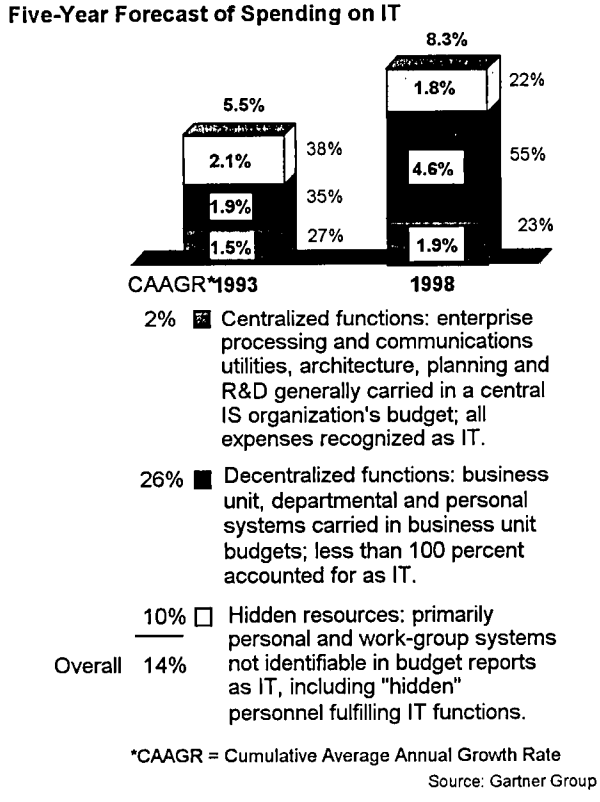


図2　今後5年間の情報技術への投資予想

とにかく，分散コンピューティングへの投資がいかに増えていくかに注目して欲しい．

■急激に向上する価格性能費

図3は，ハードウェアの価格がどんどん下がっていく傾向を表したグラフである．この4年間でMIPS当たりの単価は，毎年25～30％も下降を続けている．この図はIBM社[*2]の機械を対象に調べたものだが，どこのメーカの機械に関係なく同じ傾向となっている．メインフレーム市場でも新しい価格競争に入っている．

ヒューレット・パッカード（HP）社は，ミニコンのメーカと考えられがちだが，今やCMOSテクノロジに基づき，メインフレームのレンジに入るようなシステムの開発を行っている．

CMOSのMIPS当たり単価は，どんどん下がっており，ECLを採用した新しいメインフレームを買うと，1997年にその性能あたりの単価はゼロに等しくなってしまう．処分するのに金を払う必要があるので誰も引き取ってくれないかもしれない．

メンテナンス―保守のコストでも同じ傾向が見られる（図4参照）．

CMOS採用の新しい機械は，消費電力が少ないし，筐体も小さく，しかもプロセッサ自体も非常に小型化している．したがってメンテナンスのコストも現在の機械と比

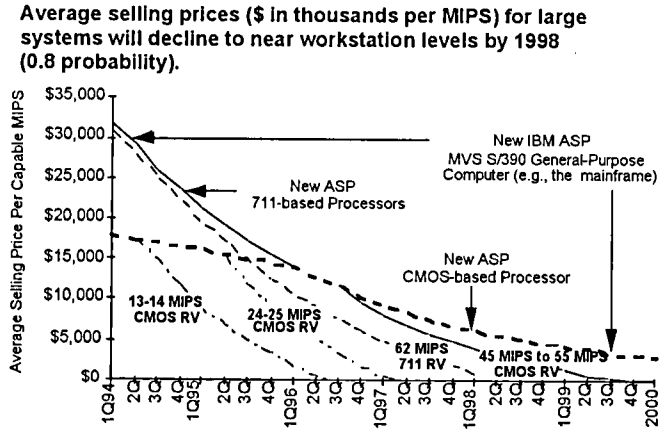


図3　ハードウェアコストの傾向

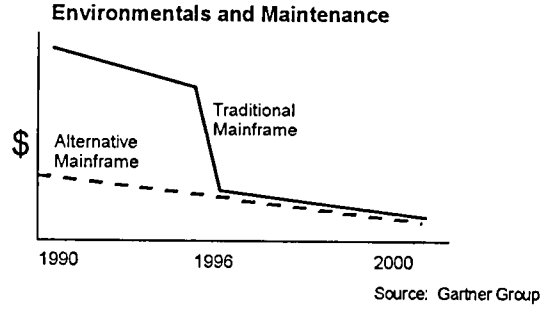


図4　メンテナンスコストの傾向

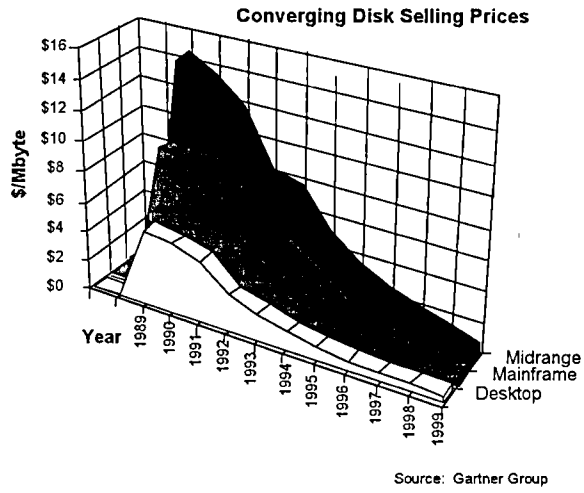


図 5　ディスクデバイスコストの傾向

べ 15～20 %削減できるだろう.

さらに，ディスク・デバイスについても同様のことが言える（図 5 参照）.

現在，ディスク・デバイスは，デスクトップ，メインフレーム，ミドルレンジ用の 3 種類が市場で提供されているが，1999 年までにこれらの価格が近寄ってきて二つにまとまることになろう.

一つは非常に安いディスク，もう一つはそれよりも少々高いもので，どちらを選択するかはシステムの機能によって選ぶことになろう.

デスクトップの場合は 1 M バイト当たり 25～30 セント，メインフレームでは 1 M バイト当たり 3～3.5 ドルくらいになろう．これは今後，年率 30 %下降していくことを示している.

■コモディティ化により変革する企業構造

このような価格競争の結果，ベンダーも利益を上げるために企業構造を変えていかなければならなくなってきた（図 6 参照）.

ベンダーは，従来のように自分の企業内で開発から製造までのすべてを行うという形から，コモディティ化された安いものを他社から購入してきて，製品化するということをやり始めた.

また，コモディティ化された戦略的な製品を自社で販売するだけなく，VAR (バリュー・アデット・リセイラー）を介して提供することも開始した.

さらに複雑な製品を扱う場合は，インハウスのシステムを作り，そのサービス・サポートの経費を請求するとか，サポートそのものを売り込むというのも一つのやり方である.

皆さんのようなユーザにとっての変化は何かと言うと，ユーザモデルが変わってくるということである．過去においては，システムをそのまま買ってくればよく，システムのインテグレーションなどを行う必要がなかった.

ところが，分散環境では自分でいろいろなシステムを統合化する必要がある．おそ

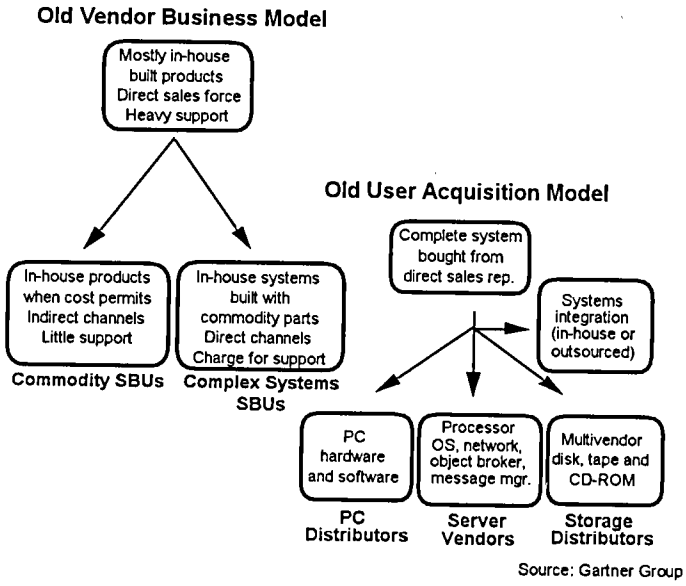


図 6　ベンダーの企業構造の変化

らく5社ほどのベンダーを相手にしなければならないだろう．すると皆さんは自分の要求に合ったベンダーの製品を選ぶという責任を持つことになる．次にその製品をテストしインストールするという責任も，メンテナンスの責任も持つことになる．5社のベンダーを使っていても，3〜4年経つとその中の1社が倒産して消えているかもしれない．そうなったとき，どう対処するかも考えておかなければならない．

このようにシステム統合化の責任は非常に大きなものとなってくる．昔は1社のベンダーの1人のセールスマンと話をしていれば，すべて要求にあった製品が買えたし，中身もわかった．

今やPCのセールスマンやオブジェクト言語のセールスマンなど，いろいろなベンダーの人と合う必要があり，仕事が非常に複雑になってきた．

また，複雑構成の機器管理のコストも高くつくようになってきた．したがって，機器の購買はできるだけ統合してベンダーの数は絞るべきだと思う．その方が価格も有利になるし，交渉もうまくいくようになる．例えば，PCは同じベンダーから，ディスク・デバイスは同じベンダーから購入する方が価格交渉でも有利に働くことになる．

■コンピュータ化の新しい傾向

表1は，コンピュータ化の新しい傾向を示したものである．このうち二つのトレンドについて触れてみる．

<Diversity and heterogeneity>というのは，多様性・異種性のことである．

今や異種で多様なシステムが存在している．基幹業務系のビジネスで使用されるメインフレームでは，いろいろなアプリケーションが幅広い分野に散らばって利用されている．そして，IT価格が下がったために過去においては夢とも思われたアプリケーションが現実的に可能になっている．特に，意思決定支援システムについて，それが言える．企業で発生したデータは記憶できるけれども使われないから，その80％以上

表 1　コンピュータ化の新しい傾向

What are the dominant trends and true costs of "new age" computing?

Trend	Implication
Diversity and heterogeneity	High costs and risks for C/S, OA and DSS
Middleware through objects	Frequent and continuous migrations
Innovation at point of least resistance	Mobile platforms dominate, end-user computing is a moving target
Architecture provides "virtual integration"	IS must be restructured, not reorganized
Competitive advantage after chaos	Ignoring TCO and financial ratios erodes value of IT
TCO is the real "budget"	TCO helps justify investment in "infratecture"

は捨てられてしまっている．ところが，意思決定支援システムのようなAPではデータ・マイニングが可能となる．つまり，これまで捨ててしまっていた大量のデータの中から金を掘り起こそうというのがデータ・マイニングであり，それが可能になってきている．また，OA分野においても従来，メモリーを大量に使用するため，コスト的に難しいと言われたイメージ処理，画像処理も簡単にできるようになってきた．

それから＜Innovation at point of least resistance＞とあるが，これは“抵抗の一番少ないところでイノベーションしよう”というもので，マイクロソフト社の戦略でもある．つまり，中央のIS部門がコントロールしていない部署向けに新しい製品を開発しようという戦略である．

Windows 95*3 は，デスクトップの製品であり，IS部門が知らない所で，勝手に使える製品である．この戦略によってマイクロソフト社は急成長したのである．

C/SSを管理するのは難しい．しかも，C/SSは，IT関連総予算の中で占める割合が大きくなってくるわけだが，これまで述べたような観点から総費用をとらえ直す必要があろう．

■ C/SSの概念

図7は，ガートナ・グループが考えたC/SSの概念図である．これまでC/SSといえば，左端のように2層構造だった．つまり，サーバとクライアントがあって，その間にコミュニケーションがあるという図式だった．しかし，それが3層構造化してくる．3層構成のC/SSアーキテクチャになってくるというわけである．

その理由は，まずトランザクションを簡略化すること，そしてアプリケーションの移植性を高めることにある．

C/SSに移行するのに，どのくらいコストがかかるのかという質問をよく受けた．というのもソフトウェア開発ツールのベンダーがやってきて「C/SS化すればコストが3分の1，あるいは2分の1で済み，しかも処理スピードも増すようになります．だから今のメインフレームを捨ててしまいなさい」と言ってくるが，果たしてそうなのかどうか迷っている人が多いからである．

しかし，実際にC/SSに移行した人の経験を聞いて見ると，悪い結果の方が多い．

例えば，アメリカ西部のあるテレコミュニケーション会社では，8億ドルを投入して

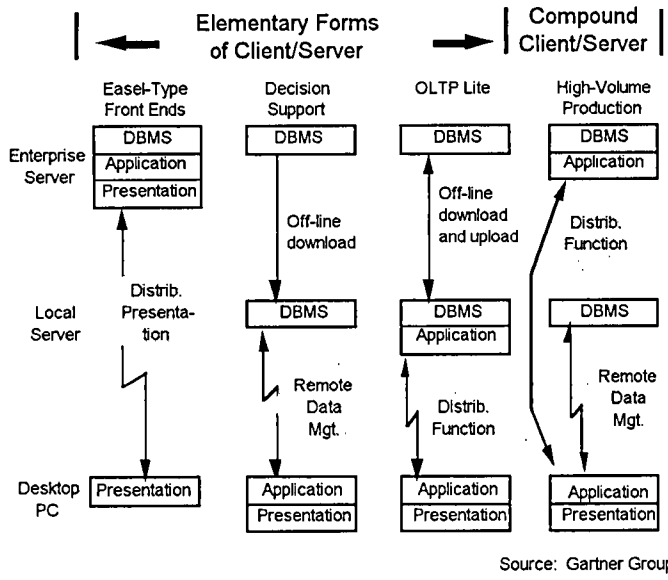


図 7　C/SS 概念図

完全分散型の C/SS を構築した．そして今年の初頭に分散化を止め，再度メインフレームに戻してしまった．メリットはそれほどでもないのにコストがあまりにもかかりすぎるというのが理由である．

■ C/SS とコストの問題

図 8 は C/SS 化にどのくらいコストがかかかるを調査した結果を円グラフで表したものである．

C/SS のクライアントが 25 ほどの小規模のものから，サーバの数が 100 を超え，ク

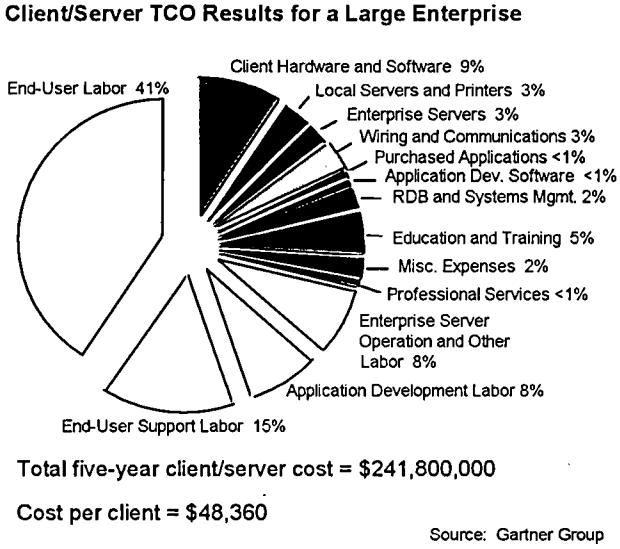


図 8　C/SS コストの調査結果

ライアントが2500に及ぶ大規模なものまで，調べたところ，これまで予想していたことと違った結果が出てきた．つまり，メインフレームでやるよりも3倍以上のコストがかかってしまったという結果が出たわけである．

分析して見ると，クライアント用のソフトウェア/ハードウェアで9％，ローカル・サーバとプリンタで3％，エンタープライズ・サーバで3％，通信回りが3％であり，これだけを足しても20％に満たない．残りの80％近くは，教育・トレーニング費やAP開発のための人件費，エンドユーザ・サポートのための人件費などの間接費で占められている．したがってC/SSを成功させるには，これらの人件費を何とか抑制する方法をとらないと，そのプロジェクトは失敗する．

また，広範囲に分散化されたシステムには，ソフトウェアの開発ツール，自動化ツールなどもまだ整備されておらず，使用可能な製品として市場に出回っていない．このため，とにかく人間をつぎ込むしかないということから，人件費が非常にかさんできているのである．

調査では25人のユーザに対し1人のサポータがいるという結果も出ている．メインフレームなら1人で130～140人のユーザをサポートできるわけだからC/SSでは6倍ものスタッフが必要になってくるのである．

■ C/SS化と情報システム部門の役割

メインフレームの中には既存のシステムがあり，エンドユーザにとって有用なものが多い．このため，既存システムをリエンジニアリングしようという動きが出ている．2～3年前，アプリケーション開発コストは高くなると言われた．しかし，情報技術のコストはどんどん下がっている．アプリケーションを簡単に他のプラットフォームでも代替処理できるインタフェース・ソフトウェアも出てくるだろう．そうしたものを使えば，メインフレームのアプリケーションがどんどん使いやすくなってくる．そして既存のシステムをリエンジニアリングして手を加えることによってさらに有効なシステムにすることが可能になる．

C/SSのアプリケーションも将来，必ずこうした方向に変わってくるはずである．したがってIS部門の人達は，それを手助けしていく必要がある．これまでIS部門が培ってきた技術は沢山あり，それは分散環境でも活用できるものである．エンドユーザの人達は，IS部門の協力が得られるということを理解すべきである．IS部門は，販売担当のマネジャーのように，その技術をエンドユーサ部門に売り込んでいく必要がある．

■情報システム部門は利用部門との協力関係を

これからはIS部門にとって重要になるのは，エンドユーザに対するサポートである．

ヘルプデスクが広まっているが，このヘルプデスクのあり方もどんどん変化している．表計算ソフトウェアを使いたいが，どうしたらいいのかといった質問に答えるだけでなく，LANの管理やディスクの管理，または従来のアプリケーション・システムに関してなどあらゆるエンドユーザの質問を受付けて答えていく姿勢が必要だろう．

図8にあるLabor，すなわち人件費の部分は，これまで低く見積りすぎていたような気がする．こうしたコストを正確に把握していなかったように思う．サーバのオペレータとしてバックアップ作業をしたり，ソフトウェアのインストレーションをしたり，ハードディスク・ドライブのフラグメンテーションをしたり，スクリーン・セーバを変えたりするのにどのくらいの時間を使っているかというと，1週間のうち5時間30分も費やしている．これは1週間の中の15〜16％の時間に相当する．だから，この作業部分をIS部門の人達がもっと引き受けるようにならないといけない．IS部門は技術のことをよく知っているから，それだけエンドユーザ部門を助けることができるのである．それとクライアントPCが3000，5000，10000台と増えていくとき，ソフトウェアのアップグレードは非常に困難になっている．各PC設置場所へ行ってディスケットを取り替える必要があるからである．そのような場合，IS部門の協力を得て，電子的に処理するEDIで配布すればコスト的にも能率的にもずいぶん助かることになる．

アメリカのシアトルにある某メーカでは7000台のPCを11の事務所に設置して利用していた．そしてEDIによるソフトウェアのディストリビューション・システムを導入した．その分コストはかかったが，7か月半で，そのコストの埋め合わせをしてしまった．というのは，それまで自動車でシアトル中を走り回って，各事務所のPCのソフトウェアのアップグレードを行っていたが，その費用は年間550万ドルも要していたという．こうしたソフトウェアのアップグレードの問題にしても電子的に処理できるようにすれば，効果的で迅速な処理が可能になるのである．それは，エンドユーザだけではできない．IS部門の助けが必要なのである．

■集中化の動きと移行の方式

そうしたことから再度，集中化の動きが見られてきた(図9参照)．これは分散システムを完全にIS部門に戻してしまうことではない．IS部門がユティリティ・サービスを行うとか，ITアーキテクチャを作ってあげるとか，エンドユーザが苦手な技術サービスや技術の評価を行うといったことである．

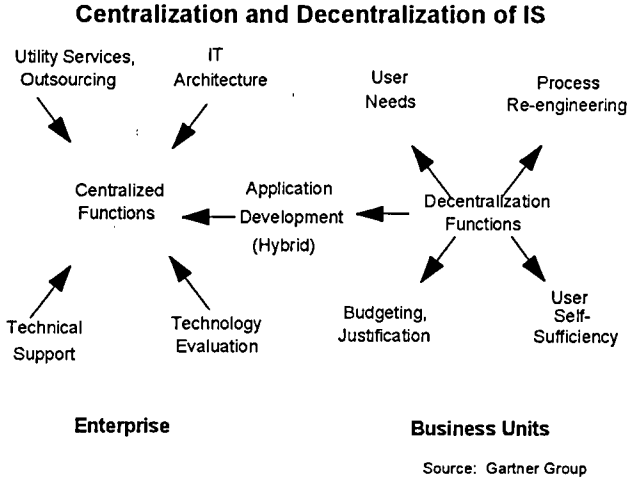


図9　情報システムの集中と分散

一方，図の左右は，各部門が持っていた方がよいと思われる機能を示している．

そして，その中間にあるアプリケーション開発が大きく変わる．APの開発は，IS部門が全てを行うのではなく各現場部門と協力してAPの開発を行う．こうしたやり方の方がユーザニーズに合ったAPの開発が行えるようになるからである．これがAP開発における一番大きな変化となる．そして，新しい環境に応じたアプリケーション・システムの移行が必要となる．このAPの移行には四つの異なる方式がある(図10参照)．

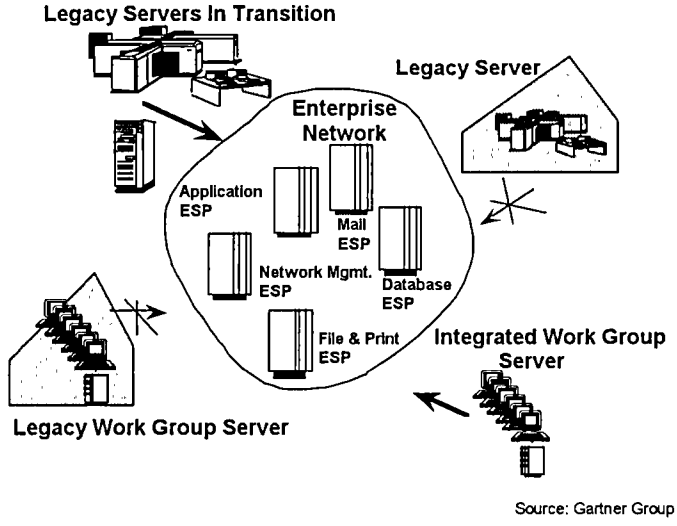


図 10　APの移行

一つは，<Agree to Disagree>と呼ばれるもので，これは新しいエンドユーザのためのシステムを別個に作る方式であり，あまり良いアプローチとはいえない．

二つ目は，<Clear and Reseed>と呼ばれるもので，これは古いものはすっかり捨ててしまって新しいプラットフォームに移してしまうやり方であるが，成功した例はあまり聞いていない．

次の二つは，現在よく行われている方式である．つまり，

三つ目は<Fade In, Fade Out>と呼ばれるもので，従来のシステムを運用しながら，環境の変化に応じて，最適なプラットフォームとITを使って，リエンジニアリングしていく方式である．

すべてのアプリケーションを一度に作り直すのではなく，優先度に従って段階的に進めていくやり方で，成功率が高いものである．

四つ目が<Surround and Supplement>と呼ばれるもので，従来のメインフレームのシステムは残しておき，その中に新しい技術，例えば意思決定で用いられるデータマイニングなどをバックエンドに付け加えたり，新しいソフトウェアなどを付加していく方法である．

■新技術の導入と人間の問題

新しい技術を導入する上で，最も大きな障壁になると考えられるのは人間である．人間は大きな変化を起こそうとする場合，必ず抵抗を示すからである．人間というものは，どうしても現状を保持したいと求めるものである．だから強制的にやらないとこれまでのやり方を変えることはできない．しかし，皆さんのIS部門もリストラクチャしなければならない．新しいビジネスの形に合わせる必要がある．

現在，メインフレーム担当の人間はメインフレームのことだけを，エンドユーザ担当の者はPCのことだけを，ネットワーク担当の者はネットワークだけのことを，UNIX担当の者はUNIXのことだけを行うグループとして構成されていることが多く，それぞれ勝手に仕事を行い，相互に交流することは少なかった．PC担当の者がメインフレーム担当の者と対話することは難しかった．まるで着ている服も，映画の趣味も，食べ物の趣味も異なるというような具合であった．しかし，C/SSプロジェクトを成功に導くためには，各グループの技能を混ぜ合わせる必要がある．そうでないと成功はおぼつかない．

メインフレーム担当者とPC担当者，それにネットワーク担当者の存在価値は異なる．だからそれぞれの総合力を発揮できるようなIS部門のチームに全体を作っていく必要がある．

ガートナ・グループとしては，環境変化に対応してどのような管理を行っていくかということで＜Change Management＞というものを提案している．しかし顧客は，人間を管理するのは難しいということからこの＜Change Management＞をコンサルタント契約からはずしたがる．

われわれの経験では，C/SSプロジェクトを進める場合，人間の問題を取り上げないと失敗する確率は非常に高くなると思っている．人間は変化に対し抵抗するが，それを無視することも必要である．

■重要性を増す情報システム部門の責務

IS部門のマネジャーの任務はますます大きくなり，チャレンジすることが多くなる．技術に対する抵抗はどうしても出てくる．エンドユーザは，2〜3年後にどのような技術が出てくるかといったことにあまり関心はない．今日の問題をどうするか，どう技術が解決してくれるか，に関心があるのである．1998年にはこうなるのだというテクノロジー・ビジョンを示しても「そんなことはどうでもいい」と考える．もし，今日の問題が解決できないとなると外部の会社に頼んでしまうかもしれない．これは皆さんにとって危険なことである．

オーストラリアの保険会社の例がある．この保険会社は1991年に，重大な意思決定を行った．つまり，組織を六つの戦略事業単位に分割した．それまで同社の情報システムは，220 MIPSのメインフレームを使ってすべてのアプリケーションを処理していた．ところが6事業単位に分割されたとき，各事業部門長に「従来のメインフレームを継続して使用するか，あるいはシステム・パッケージを買ってきて利用するか，独自にハードウェアを購入して新規にアプリケーションを開発するか」といった選択権が与えられた．その結果，6人のマネジャーは全員，それぞれ新しいアプリケーショ

ンを開発することに決めた．これは1991年のことである．

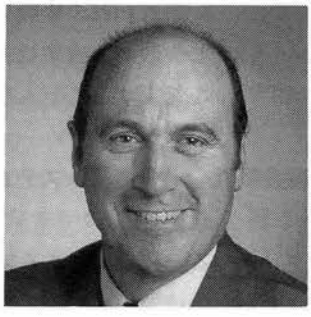
こうなると集中処理に使用されてきたデータセンタの220 MIPSのメインフレームは何もすることができない．結局，本社の電子メール・システムに使用されることになった．それだけである．同時に，IS部門で仕事を行っていた人達もやる仕事がなくなったので大勢が辞めてしまった．1991年から1993年にかけて各事業部門はそれぞれ独自のシステム開発に取り組んできたが，93年の中頃になっても全然，進展がみられない．開発スケジュールが非常に遅れてしまったのである．しかも，当初考えていた予算よりも2倍も投資してしまった．結局，難しすぎてどうにもならない．お手上げだということになり，再度，集中化のシステムに移行することになった．そして，IS部門へ助力を求めてきたが，IS部門では人が辞めてしまっていて人材はいない．このため，1からIS部門を作り上げることになったのである．しかも，各事業部のマネジャーは，四つの異なるプラットフォームを買ってしまった．三つの異なるデータベース，六つの異なるアプリケーション環境が残され，IS部門は，それらをすべて取り扱う戦略を考えなければいけなかった．

こうした状況に皆さんの会社も陥らないためにエンドユーザに対して明確な方向づけを示す必要がある．これからの5年間は，情報技術の変革も激しいので，はっきりとした方向づけをしておくことが重要だと思われる．

＊1　UNIX：X/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
＊2　IBM：International Business Machines社の登録商標である．
＊3　Windows：米国Microsoft社の登録商標である．Windows 95は同社の製品名である．

執筆者紹介　クリス・モリス（Chris Morris）

複数のコンピュータ・ベンダーにおいて，マネジメント，マーケティング，システム・エンジニアなどを担当した後，1992年にガートナ・グループに入社．現在，アジア・オセアニア地域の顧客に対して，情報技術市場の動向分析およびコンピュータ・システムの戦略的導入に関するアドバイザリー・サービスを提供している．現在，アジア・パシフィック・リサーチ・センターにおいてプログラム・ディレクタ．



新世代メインフレームの基本思想

Basic Concept of New Generation Mainframes

今 江　泰, 丑久保　年常

要 約　オープン・システムを中心とするC/Sシステムの実稼働が増加するにつれて，メインフレームや集中処理型システムの見直し気運が出てきている．今後の企業情報システムの形態を新世代メインフレームとオープン・システムとがそれぞれの特徴を生かしつつ共存/連携するエンタープライズ・クライアント/サーバ・システムととらえ，そこでのエンタープライズ・サーバとしての新世代メインフレームの位置づけ，有効性，求められる要件，要件を実現する要素技術について述べる．

Abstract　With the growing utilization of client/server systems running mainly on open systems, the trend has emerged to re-evaluate mainframes and centralized processing systems. Future corporate information systems can be conceived as enterprise client/server systems providing the synergism between the new generation mainframes and open systems by utilizing the various features for coexistensive and cooperative processing. This paper describes the role of these new generation mainframes used as enterprise servers, their effectiveness, the requirements, and the key technology to satisfy those requirements.

1. はじめに

バブル崩壊後の不況の中，コスト削減が企業の最優先課題となり，情報化投資にも大きな影響を与えた．折りから不況が先行した米国で注目を集めたオープン化，ダウンサイジングの波が日本にも到達した．メインフレームの終焉が話題になり，価格低下の著しいUNIX*1機やPCなどのオープン・システムを使用したクライアント/サーバ(C/S)システムの導入が活発に行われるようになった．このシステムは，ユーザの操作性の良さや柔軟性などの特徴を持っている一方で，運用管理上の問題や基幹系業務システムに求められる安定性やデータの保全性に課題がある．また，既存のメインフレームの業務をダウンサイジングすることの困難さも次第に認識されるようになってきた．

このような流れの中で，既存資産を有効に利用しながら，オープン・システムとメインフレームのそれぞれの特性を生かした適材適所型の企業レベルのC/Sシステムへ向けて，なだらかな移行を進めていくことが見直されている．

本稿では，このような状況を受けて，これからの企業情報システムにおけるメインフレームの有効性に焦点を当てて述べる．2章では新世代の企業情報システムのモデルとそこでのエンタープライズ・サーバとしてのメインフレームの位置づけを，3章ではエンタープライズ・サーバに求められる基本要件とそれを実現する技術について最新鋭機ITASCA 3800シリーズを例に述べる．

2. 企業情報システムと新世代メインフレームの位置づけ

2.1 背　　景

2.1.1 ビジネス環境の変化

企業を取り巻くビジネス環境は急激な変化の時代を迎えており，この状況の中で競争優位を確保し維持していくためには，変化に対する企業の素早い対応が必須となっている．したがって，情報システムに求められる要求も大きく変化してきている．具体的には，業務系を中心とする高速性，大容量，信頼性に加えて，

- ●変化に対応しやすい柔軟性
- ●データ/情報への容易なアクセス
- ●業務プロセスのコンピュータ化
- ●新技術の採用（例：GUI 機能，クライアント/サーバ技術，等）

などの新しいニーズが発生している．

2.1.2 コンピューティング・パラダイムの変化

一方，情報・通信技術の進展に伴ってコンピューティング・パラダイムも大きく変化しつつある．以前はメインフレーム中心の集中処理形態が大部分であったが，その後，UNIX 機や PC が普及し始めクライアント/サーバ（C/S）モデルに基づくシステム形態も現れた（第 1 世代の C/S システム）．最近の PC の高性能化/低価格化には著しいものがあり，LAN やインターネットの急速な普及と共にコンピューティング・パラダイムの変化に大きな影響を与えつつある．

メインフレームも CMOS 化によって価格性能比が著しく向上し，オープンシステムとの連携機能も大幅に強化したことにより，これをエンタープライズ・サーバと位置づけ，PC や UNIX 機を統合したエンタープライズ・クライアント/サーバ・システム形態（第二世代の C/S システム）が今後の主流になっていくと考えられる．

当社では，1990 年に発表した 90 年代のソフトウェア体系である UA（Unisys Architecture）で，すでにこのようなコンピューティング環境をインフォメーション・ネットワークと名づけ，90 年代に求められるシステム像として提案した．最近，市場では，インフォメーション・ネットワークを意味する語としてネットワーク・セントリックという用語が使われ始めた．

ユーザにとっては，このようなエンタープライズ・クライアント/サーバ・コンピューティング環境へどのように移行していくかが問題となる．当社では，新世代の企業情報システムのあるべき姿とそこへ至る移行過程の考え方を ClearPath[*2] として提唱し，さらにこれを実現するプロダクトとサービスを ClearPath ソリューションセットとして提供している．

2.2 企業情報システム・モデル

2.2.1 企業情報システムに求められる要件

新世代の企業情報システムは，IT（情報技術）の活用によって 2.1 節で述べた要求に応え，企業力を強化するものでなければならない．つまり，次の要件を満足する必要がある．

- ●企業内のみならず取引先，関連企業，個人顧客を含めた企業外との情報の交換と共有による顧客サービスの向上

●売上拡大やマーケット拡大につながる戦略的な情報システムの構築
●ビジネス・プロセスあるいは外部環境の変化に即応できる基幹業務システム
●データ/情報/知識の共有化によるオフィスの生産性向上
●複雑化している分散処理環境の統合など効果的/効率的なIT資産の管理・運用
●情報システム全体のコスト（TCO＝Total Cost of Ownership）の削減

図1は，新世代の企業情報システムのイメージ図である．企業レベルのエンタープライズ・サーバ，部門レベルの部門サーバ，チーム/個人のワークステーションやPCが互いに連携し，さらに企業外ともインターネットなどを経由して情報の交換や共有が可能なシステムである．このようなシステムでは，個人のPCからチーム/部門/企業レベルの各サーバそして企業の外や海外のサーバまでも対象にした情報の交換や共有がシームレスな環境で実現できる．つまり，PCの前にいるエンドユーザには，ネットワーク全体が一つのシステムに見え（シングル・イメージ），さまざまな業務を実行すること（マルチ・パーパス）が可能になる．

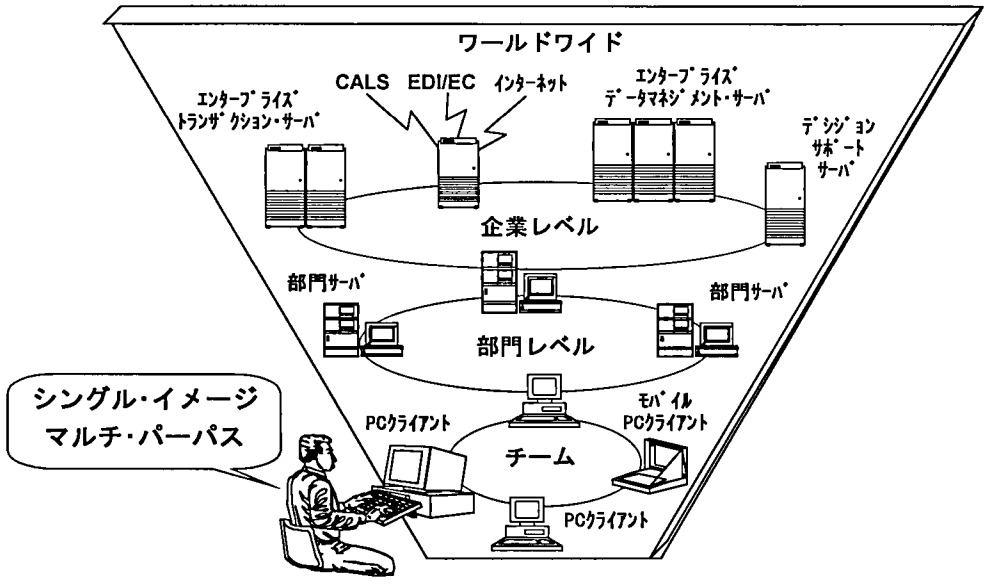


図1　新世代の企業情報システム

2.2.2　企業情報システム・モデル

当社が定義した新世代の企業情報システム・モデルについて述べる．このモデルでは企業情報システムを，業務系，情報系，オフィス支援系，連携基盤の四つのドメインに分類している（図2）．初めの三つはアプリケーションの分類であり，残りの連携基盤はこれらのアプリケーションに共通の分野である．

1)　業務系

企業活動の基幹となる業務で，そのシステムが停止すると企業活動そのものが停止するミッション・クリティカルな分野が対象である．アプリケーションの例としては，勘定系システムや予約システム，受発注システムなどがある．

2)　情報系

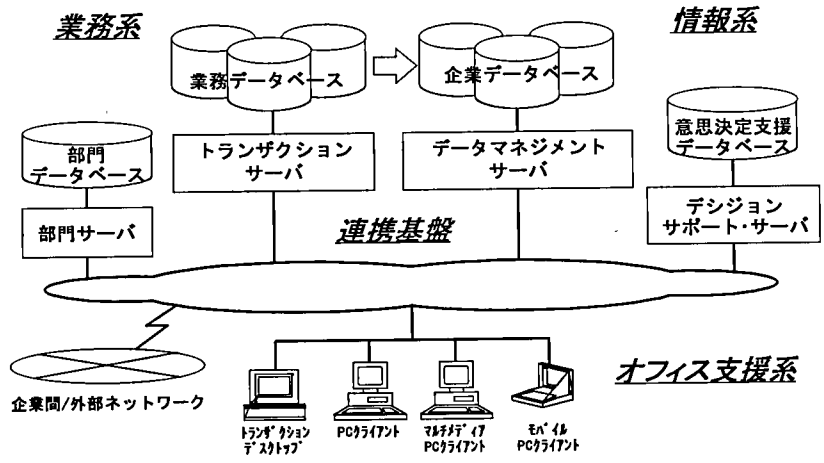


図 2　企業情報システム・モデル

業務系で蓄積されたデータを元に検索・分析を行い，企業戦略や戦術の立案に活用する分野である．情報活用，知的業務支援の観点から現在，最も注目されている分野である．企業レベルのすべてのデータを蓄積・管理し利用する形態と，目的別に抽出した大量データの高速検索を行い意思決定を支援する形態がある．アプリケーションの例としては，経営分析，市場分析・開拓，顧客サービスなどがある．

3)　オフィス支援系

個人/チームから企業レベルに至るまでグループ協調作業を支援する分野で，ホワイトカラーの生産性向上の観点からも近年注目されている．利用される技術としては，電子メール，グループウェア，ワークフロー管理，イントラネットなどがある．当社の企業情報システム・モデルでは，取引先，関連企業，顧客など外部とのデータや情報の交換や共有もオフィス支援系に取り込んでいる．

4)　連携基盤

企業内および企業の外とのデータ，情報，知識の交換と共有を可能とするネットワーキングを実現する分野である．上記の三つのアプリケーション分野に共通の基盤を提供する．

2.3　企業情報システム・モデルにおける新世代メインフレームの位置づけ

2.3.1　エンタープライズ・サーバとしての新世代メインフレーム

メインフレームは従来からの特徴である堅牢性，可用性，信頼性，データの保全性に加え，オープン・システムとの連携機能，つまり，インターオペラビリティやポータビリティの強化によって，エンタープライズ・サーバとして新たな役割を担う．柔軟性と操作性を生かすことによって，顧客サービス，生産性，サービス品質の向上が可能となるクライアント/サーバ環境で，新世代メインフレームをエンタープライズ・サーバとして利用することによって適材適所型のいわゆるライトサイジング・システムの構築が可能になる．

2.3.2 業務系における新世代メインフレーム

1) 業務系の特性

業務系は企業活動の基盤となる業務で，その業務が停止すると企業活動そのものが停止する恐れのある業務である．業務系システムの目的は，日常業務の基本的な処理の自動化である．主な機能としては，定型的なトランザクション処理の実行で商品情報や顧客情報の生成，蓄積，更新を行い，帳票の発行なども行う．

主なアプリケーションの例としては，勘定系システム，座席予約システム，受発注システム，生産管理システムなどの基幹業務アプリケーションと人事システムや経理システムのような業務支援アプリケーションがある．

基幹業務系では，障害/災害に対する高信頼性，高い処理効率(スループット)，24時間365日無停止運用のような処理特性を要求される．

2) エンタープライズ・トランザクション・サーバ

基幹業務系の中核を担うサーバは，エンタープライズ・トランザクション・サーバである．このサーバには，安定性，高信頼性，大規模な拡張性が不可欠である．また，処理形態はトランザクション処理とバッチ処理が中心となる．

新世代メインフレームは，このエンタープライズ・トランザクション・サーバとして最適である．

2.3.3 情報系における新世代メインフレーム

1) 情報系の特性

情報系は，業務系で発生した生データを情報として利用可能になるように抽出・変換・蓄積し，検索・加工・分析・予測・統計解析・シミュレーションなどの計画や意思決定を行う業務である．情報系の目的は情報の活用による企業競争力の強化である．主な機能としては，業務系で発生したデータや市場情報などの外部情報を加工し利用することを可能にしたり，エンドユーザに検索と分析の機能を提供することである．

主なアプリケーションとしては，経営者支援システム，営業支援システム，管理者支援システム，データベース・マーケティングなどがある．

情報系では，大規模データベース処理，大量データ・アクセス，高速検索などの処理特性が求められる．

2) エンタープライズ・データマネジメント・サーバとデシジョンサポート・サーバ

情報系のサーバはその用途によって二つの形態に分類できる．一つは企業レベルの統合的なデータベースを保持するエンタープライズ・データマネジメント・サーバであり，もう一つは目的別のデータベースを保持する高速検索専用のデシジョンサポート・サーバである．

エンタープライズ・データマネジメント・サーバには，大規模な拡張性，高信頼性，高速検索機能などが求められる．並列I/O機能や高速入出力処理をサポートする新世代メインフレームは，エンタープライズ・データマネジメント・サーバとして最適である．エンタープライズ・データマネジメント・サーバにはPC上の市販ソフトウェアからのアクセスに対応する機能，例えば，マイクロソフト社

が定義したODBC（＝Open Database Connectivity）*3 機能のサポートなども必須である．

一方，デシジョンサポート・サーバとしては，大量データの高速検索に向く超並列機が有効であり，当社ではOPUS 2000を提供している．

2.3.4 オフィス支援系における新世代メインフレーム

1) オフィス支援系の特性

オフィス支援系は，情報の交換と共有のための連携基盤を活用し，情報の保管・共用・再利用・配布・伝達等を行うことによって，各種業務の補完や支援を行う分野であり，オフィスワークの生産性向上を目的とする．企業活動の真の生産性向上のためには，個人/チーム・レベル，部門レベル，企業レベルを越え，CALS（生産・調達・運用支援統合情報システム）やEC（電子商取引）による取引先，関連企業，顧客など外部も含めたデータや情報の交換・共有も重要であり，今後，この分野の重要性は増大する．

2) 機能別サーバ

従来，オフィス支援系ではPCやUNIX機が主に使用されており，今後もこの傾向は変わらないと思われるが，システムの規模が部門レベルから全社レベルへと拡大するにつれ，システムの安定性や大量データの保管機能が重要視されるようになってくる．この場合，堅牢性やデータ保全性を特徴とし，さらに小型化・低価格化を実現した新世代メインフレームを機能別サーバとして効果的に利用することも可能になる．例えば，企業内に分散配置されたPCやUNIX機のデータの保全は大きな問題であるが，新世代メインフレームをこれらが持つデータやプログラムのバックアップ装置として使用することが可能である．また，UNIXサーバやPCサーバと組み合わせた新世代メインフレームを電子メールや文書管理システムの大規模データ保管庫として利用し，サーバを集約することによって運用管理の負担を軽減することも可能である．当社では，このような機能別サーバに向いた次世代メインフレームの開発も進めている．

2.4 ClearPath—新世代企業情報システムへの移行—

2.4.1 ClearPath

新世代の企業情報システムは，オープンな環境の中で企業の全領域のクライアント機やサーバ機が連携し協調して稼働するエンタープライズ・クライアント/サーバ・コンピューティング環境である．ClearPathは，メインフレームの既存資産を継承しながらモダナイズ，インテグレート，リエンジニアの三つのステップを経て，エンタープライズ・クライアント/サーバ・コンピューティング環境へ進化させていく方法である（図3）．モダナイズは既存の資産に最新のITを適用し，オープンで使いやすいシステムにすることによって，既存資産を維持すると同時に拡張性を高めるものである．インテグレートは，異なるシステム（ハードウエア，アプリケーション，管理システム）を一つの環境に統合することにより，システム全体の付加価値を高めるものである．リエンジニアは，既存システムを新しいビジネス環境に適合したシステムに再構築することである．

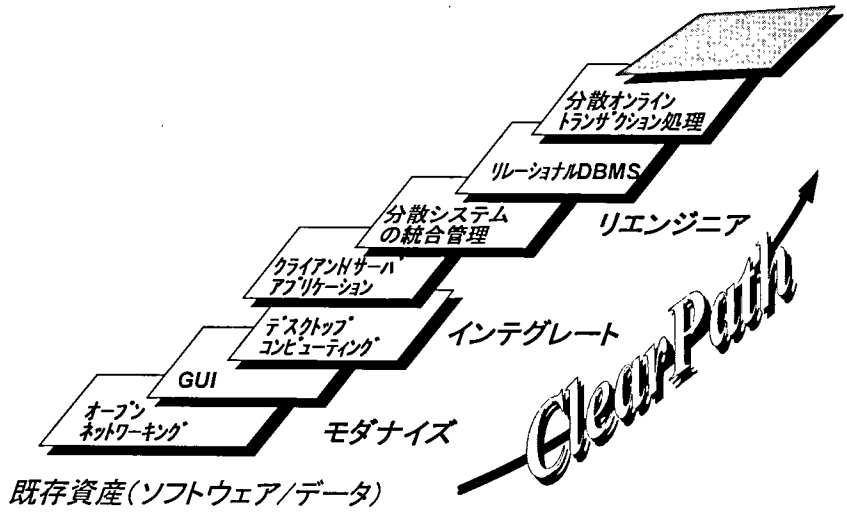


図 3　ClearPath

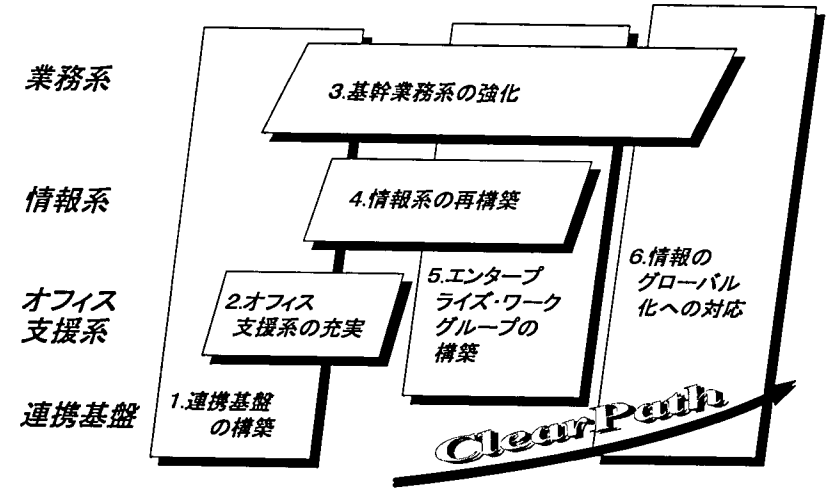


図 4　ClearPath ストーリ

2.4.2　ClearPath ストーリ

既存のメインフレームを中心とする企業情報システムを，ClearPath の考え方に基づき，エンタープライズ・クライアント/サーバ・コンピューティング環境へ段階的に進化させていく典型的な構築過程として当社が提示しているのが ClearPath ストーリである．

ClearPath ストーリは次のような大きな流れで構成される（図 4）．

① ネットワークを LAN（TCP/IP）化し，PC を導入することによって情報処理基盤を構築する．

② 電子メールやグループウェアの導入によってオフィス支援系の充実を図る．

③ 基幹業務系は，当面，既存のアプリケーション・プログラムはそのままにして，ユーザ・インタフェースのGUI化や新規アプリケーションとの連携などで強化を図る．

④ 情報系については，既存ベータベースに対するPCからのアクセスを可能にし，次に，情報活用のために統合的な企業データベースの構築やデシジョン・サポート・システム（DSS）の構築を行う．

⑤ エンド・ユーザとの接点になるオフィス支援系と業務系および情報系との連携を強化する．インターネットによる外部との情報の交換や共有も実現する．

⑥ CALSやECの導入により企業間での業務の効率化を図る．

2.4.3 ClearPath ソリューション・セット

当社はClearPathに基づき，企業情報システムを進化させていくための具体的なプロダクトとサービスをClearPathソリューションセットとして提供している．ClearPathソリューションセットは，次のような観点で体系化されている．

■企業情報システムの四つのドメインの観点

企業情報システム・モデルの定義に従って，業務系・情報系・オフィス支援系・連携基盤の各ドメインごとの進化の過程をClearPathソリューションセットとして整理した．これらは企業情報システムの主に実行環境に相当する分野である．図5は，業務系トランザクション処理にモダナイズを適用し，クライアントPCの画面をGUI（Graphical User Interface）化する例と，ネットワークをLAN化

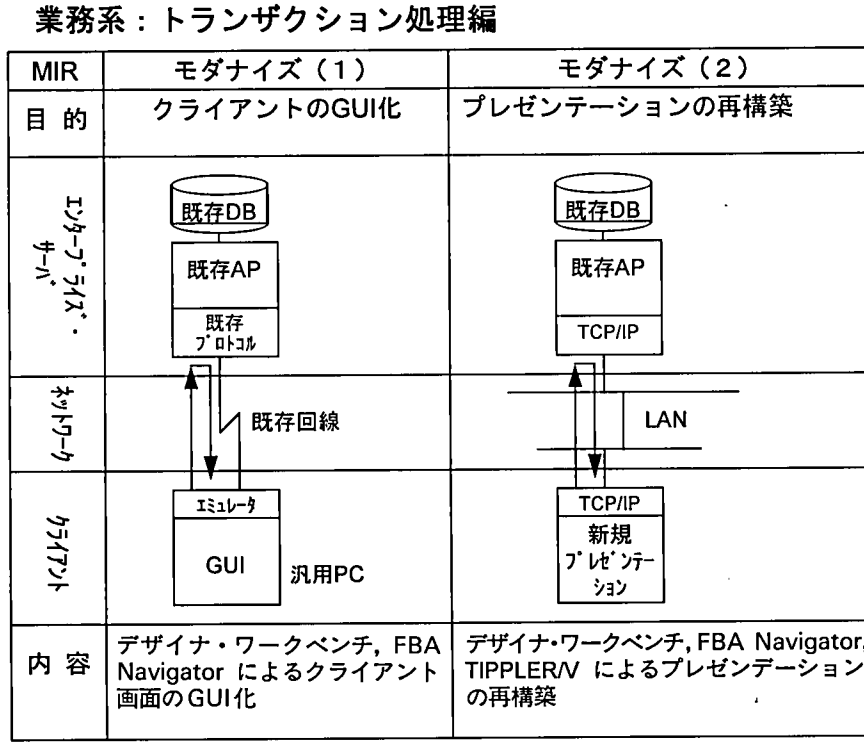


図 5 ClearPath ソリューションセットの例

し，さらにPC側にアプリケーションを組み込みユーザ・インタフェースを再構築する例である．

■企業情報システムの運用と開発の観点

システム運用管理と開発環境に対する進化をClearPathソリューションセットとして整理した．これらは企業情報システムの開発環境と運用環境に相当する分野である．

当社は，ClearPathを具現化する主要なソフトウエアを，UAを開発・実行・運用の観点から体系化した三つのフレームワークであるACCF（Advanced Cooperative Computing Framework），ASDF（Advanced Solution Development Framework），ASMF（Advanced System Management Framework）に基づいて提供している．

また，ClearPathを具現化するためのサービスを，当社のサービス体系であるUSEFUL/SVに基づいて提供している．

3. エンタープライズ・サーバ

3.1 その役割ーメインフレームからエンタープライズ・サーバへ

メインフレームの今後の役割を言及する前に，2章でも触れたがコンピュータ・システムの変遷を辿り，それを取りまく現状を見てみたい．メインフレームの今後の役割を考察する上で，重要なポイントと考えるからである．コンピュータ・システムは1950年代のバッチ処理システムに始まり，以下に示す変遷を経て今日のクライアント・サーバ・システム（C/Sシステム）の到来を迎えることとなる．

■1960年代ーTSS[1]

コンピュータの性能向上，オペレーティングシステム（OS）技術の向上，および利用者の増大を背景としてタイムシェアリングシステム（TSS：Time Sharing System-時分割システム）が実用化される．多くの端末を通信回線を介してホストコンピュータに接続し，複数の利用者があたかも同時にコンピュータを占有しているかのように使える技術である．

■1970～1980年代ー分散処理システム

TSSの普及・拡大に伴い，増大する負荷に対応するためにホストコンピュータ同士を通信ネットワークで結び負荷の分散を図った（1970年代ー水平方向への展開）．1980年代に入り，インテリジェント端末の登場により端末側での作業が可能となり，ホストコンピュータの負荷軽減のために端末側に一部の作業を分担させる垂直型のシステムが実現された（垂直方向への展開）．

■1990年代ーC/Sシステム

1980年代の垂直型の分散処理システムは，ソフトウェア技術の向上と相まって，C/Sシステムへと発展していく．ガートナ・グループの分類によれば，ユーザ・インタフェース部分（プレゼンテーション部分）と実際の処理部分（アプリケーション，およびデータベース・マネージメント部分）とに分け，前者をクライアント・コンピュータ（後者の一部を含むこともある）に，後者をサーバ・コンピュータに分担させる方式である．

C/Sシステムはシステムによっては，従来のメインフレームの代替として安価なサ

ーバで置き換えることができることもあり，情報投資軽減の意味から，また不況の追い討ちと相まって企業経営者から歓迎され，『ダウンサイジング』という言葉と共に，一部マスコミでメインフレームの終焉と喧伝された．しかしながら，ダウンサイジングに移行したものの，結果的に失敗したという例が少なくない．理由としては，

- ●ハードウェアの初期コストは安いものの，その後の運用経費が増大し，トータルでは費用が増えてしまう．
- ●設計上は十分であるが，実際には大型コンピュータなみの処理性能が出ない．
- ●システム構築にエンドユーザが関わることができるようになり，システムが複雑化して混乱するようになった．
- ●システム維持のためのエンドユーザ教育・支援の負担が大きい．
- ●システム開発に予想外の時間がかかる[2]．

の問題が挙げられる．結局のところコスト削減を目的とした性急なダウンサイジングは禁物ということであろう．また，メインフレームの終焉という考え方も短絡的すぎる．

以上のコンピュータ・システムの変遷と現状を踏まえ，今後のメインフレームの役割を考えてみたい．C/S システム自体は今後定着し，グローバルなネットワークを基盤として次の展開へと進化するものと考えられる．そしてメインフレームもこの C/S システムのサーバの一つとして組み込まれ，共存・協調していくものと考える．理由としては，

① 堅牢性，信頼性，高速・大量処理など，基幹業務を実行する上で必須となる特性を持つプラットフォームはメインフレームが最適である．

② ビジネス変化に即応するための情報システム作りが必要な今日，データ，情報，知識の全社的な共有の観点から，全社的な統合データベース・サーバが不可欠である．

③ メインフレームでも C/S システム環境に必要なオープン化が進められており，標準的なインタフェース，プロトコル，およびモデルの採用，さらには UNIX 環境の実現など，他プラットフォームとの連携基盤が確立されている．

④ ガートナ・グループによると，米国の C/S システムではデータの半分以上が情報システム部門の管理外にある．これが運用経費を増大させる一因であるが，データ管理，システム運用管理はメインフレームのように集中管理した方が効率がよい．

⑤ メインフレーム上で構築されたソフトウェア資産は膨大であり，まずはメインフレーム後継機で置き換え，少しずつオープン化に移行することが現実的である．

⑥ CMOS 技術に代表されるコモディティ技術の恩恵はクライアントだけのものではない．メインフレームでも価格性能比が向上していく．

が挙げられる．いわば単なる C/S システムではなく，企業情報システムの中核となるサーバ，すなわちエンタープライズ・サーバを機軸にしたエンタープライズ C/S システムが今後の方向性と考える．したがって，メインフレーム（以後，この章ではエンタープライズ・サーバと記述）の役割はここにある．具体的には，上記の①～⑤であ

ろう．

3.2 基 本 要 件

エンタープライズ・サーバの基本要件は，3.1節の『その役割』から自ずと導き出される．企業情報システムの中核をなすオープン・エンタープライズ・サーバとしての位置付けから，既存の基幹業務処理，大規模トランザクション処理やそれに伴うバッチ処理のための要件，データ共有のための全社統合データベース・サーバとしての要件，またネットワークを介して接続され，協調し合う多種システムの情報中枢としての要件が求められる．以下に基本要件を挙げる．

① 高速・大量処理
② 高信頼性
③ 無停止連続運転
④ 拡張性
⑤ 大規模なシステムとネットワークのサポート
⑥ オープンな相互接続性とソフトウェア資産の継承
⑦ 高生産性アプリケーション開発/実行環境
⑧ 高度なセキュリティ
⑨ 運用管理の効率化（システムの無人運転/統合管理）
⑩ 先進的なデータベース・システム

3.3 基本要件実現のための技術—並列処理技術

エンタープライズ・サーバの役割を全うするためには，高速・大量処理，高信頼性，システムの柔軟な拡張性，無停止運転，オープン・システムとの融合，ソフトウェア資産の継承，および価格性能比の向上などの要件を満足する必要がある．1995～6年に発表された各ベンダーのエンタープライズ・サーバはこの点に着眼して設計され，次の三つのキーワードに要約できる特徴を持つ．

- 密結合CPU構成クラスタを結合装置を介して複数接続した並列処理システム
- オープン環境との共存・協調の実現
- CMOS技術採用による価格性能比の向上

このうち，並列処理技術，オープン環境との共存・協調について触れたい．

3.3.1 並列処理技術

当社はいち早く並列処理システムの有用性に着目して実現に向けて対応した．この対応が複数の密結合クラスタによる並列処理をベースとした拡張トランザクション処理アーキテクチャ（XTPA：eXtended Transaction Processing Architecture）として結実することとなった．1989年にエアーラインの座席予約システムに適用されて以来，金融，電力などの本番業務で稼働中である．

3.3.2 並列処理技術の採用と強化ポイント

1) 並列処理技術の採用

トランザクション処理能力（システム・スループット），つまり単位時間当たりのトランザクション処理件数は『(1/滞留時間)×多重度』で表される．したがって，システム・スループットを向上させるためには，

●滞留時間の短縮

●多重度を上げる

ことが必要になる．一つのクラスタ内でこれらを実現しようとして，CPU の増設をしてもメモリへのアクセスで競合があるため飽和点があり，それ以上能力が向上しない．

また，単体 CPU の能力を向上させるとしても経済的な限界がある．さらに，基幹業務処理に求められる無停止という要件を加味すると，当然の帰結としてクラスタの横への展開，すなわち並列処理となる．XTPA の登場である．

2) 強化のポイント

XTPA の主な強化ポイントは，次の二つである．

●並列処理によるトランザクション処理能力の向上

●ノンストップ・システムの構築

① 並列処理によるトランザクション処理能力の向上

図 6[3] は実際のトランザクションの滞留時間の内訳である．通信処理，資源のロック，データベースのアクセス，業務処理，データベースの更新，オーディット(更新履歴)処理，そして再び通信処理に分けられる．CPU 処理時間は CPU 処理能力の向上とメモリ空間の拡大で大幅に短縮し，全体の約 3 分の 1 程度である．むしろデータベースのアクセス/更新，オーディットへの書き込みなどの入出力処理時間，および資源共有による資源ロック待ち時間の割合が無視できない．

したがって，XTPA では次の三点がポイントとなる．

●高速な排他制御（資源ロックの高速化）

●複数クラスタからの入出力競合にも耐え得る高速な入出力装置

●多重度を上げるためのクラスタ（ホスト）間の負荷分散

これを実現するのが『拡張データ処理装置（XPC: eXtended Processing Complex)』であり，以下の機能を有する．

●高速レコード・ロック制御機能

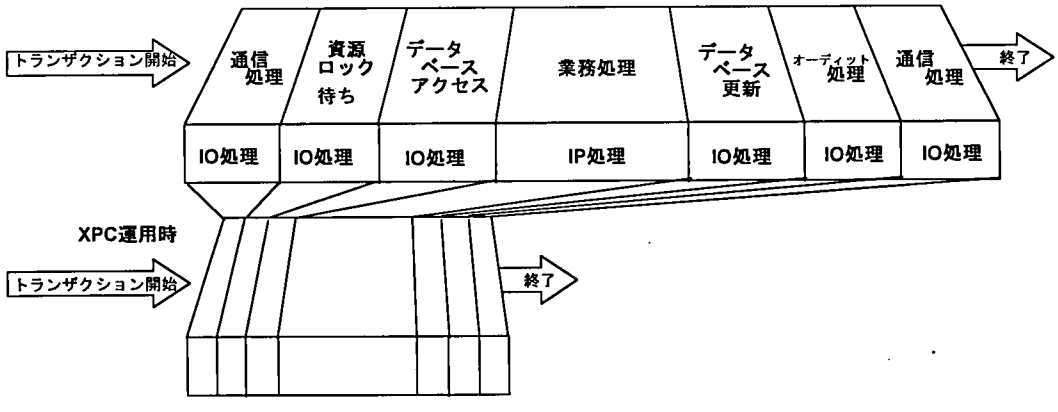


図 6 トランザクション処理のターンアラウンド・タイム

●データ入出力処理を高速化するグローバル・キャッシュ機能

●ホスト間通信機能

図7にはXPCを導入したあるユーザのXPCの効果が示されている．このユーザでは既存のユーザ・アプリケーションを変更することなくXPCを適用したが，平均TPS処理時間をXPC導入前後で比較すると，XPCのキャッシング機能によりI/Oタイム（入出力時間）が大幅に削減されていることがわかる．

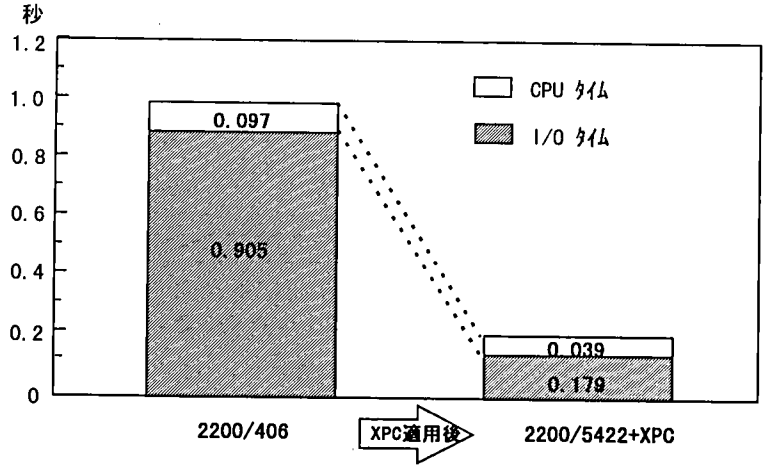


図7　あるユーザでのXPC適用効果（トランザクション処理）

② ノンストップ・システムの実現

XTPAでは並列処理による性能向上に加え，ノンストップ・システムの構築も可能である．XTPAシステムでは，各ホスト（クラスタ）は結合機構であるXPCを介して接続された構成となる．XTPAシステムの各ホストはXPCのホスト間通信機能を経由してハートビート（相互監視）を行い，ホスト障害を相互に監視している．

万一，ホスト障害が発生しハートビートが途切れた場合，必要に応じてそのホストを強制停止させると共に，自動的にリカバリ指示を出し別のホストで回復処理を行う．当然ながら，ディスク等のシステム構成要素の障害時には，その部分を本体から切り離して処理を続行する自動縮退運転機能や入出力処理チャネルの代替経路の変更などの障害対応設計も図られている．

3.3.3 並列処理を実現するハードウェア

ハードウェア群は次のものであるが，処理能力の更なる向上のため，並列処理の総CPU数とXPCメモリの拡大をしている．

① ホスト

エンタープライズ・サーバITASCA 3800，および2200/500，900．

ITASCA 3800では，従来の32CPUの倍の64CPUによる並列処理が可能．

② 結合機構

拡張データ処理装置（XPC）．XPCメモリを3.7GBから7.4GBに拡大．

③ 通信制御装置

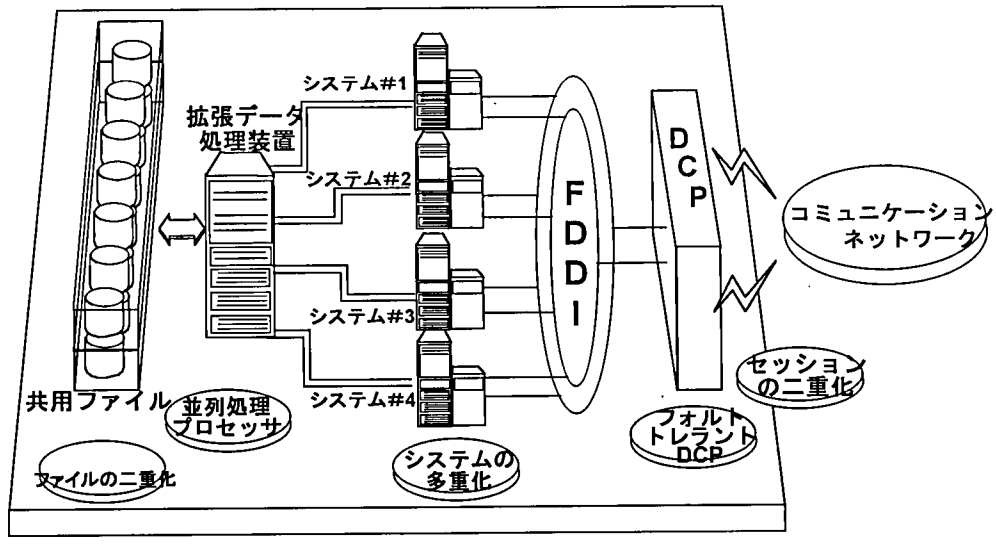


図 8　XTPA によるノンストップ・システムの実現

フォールト・トレラント・ネットワーク・プロセッサ DCP/600 FT

④　シングル・ポイント・オペレーション

複数の並列処理ホストの運用（コンソール制御，イベント制御など）をワンポイントで統合管理する SPO（Single Point Operations）

図 8 が当社の並列処理の概念図であり，基本コンセプトの一つである冗長性構成によるノンストップ・システムの実現が図示されている．

3.3.4　並列処理を実現するソフトウェア

OS 2200 と，その上で動作する次の並列処理支援ミドルウェア群を提供している．

①　データベース制御ソフトウェア

並列処理の核となるデータベース制御ソフトウェア XTC-TIP/UDS II．

②　XPC 用ソフトウェア

データの入出力を高速化するキャッシング機能を支援する VSM（Virtual Storage Manager），および複数ホストからのロック要求を並列処理する機能を支援する DSM（Distributed System Manager）．

③　トランザクション処理の負荷分散

並列処理環境にある各ホストの CPU 使用率によりトランザクションを各ホストに振り分けるバランス・マネージャ．

④　ホスト間で時間を同期させるユーティリティ

各ホストの時間を自動的に同期させる UTC（Universal Time Coordination）．

⑤　シングル・ポイント・オペレーション

複数システムのシングルビューを実現する SPO．

⑥　並列トランザクション処理での実行環境の提供

XTPA（並列環境での拡張トランザクション処理アーキテクチャ）での実行

環境を提供する XIS（eXtended Information System）.

⑦ 並列処理環境での開発・運用支援ソフトウェア

並列処理環境でのバッチ処理の効率稼働，障害対応，自動運転，障害管理等の運用を支援する IOF（Integrated Operating Facility），および並列処理環境にも対応できるリポジトリをベースとした統合開発環境支援ソフトウェア IDES（Integrated Development Environment support System）.

3.3.5 拡張データ処理装置（XPC：eXtended Processing Complex）

拡張トランザクション処理アーキテクチャ（XTPA）の中核となるのが XPC である．XPC は，二つの RISC プロセッサからなるプロセッサ演算エンジン（XIP）とホスト・インタフェース・プロセッサ（HIP），およびミラー・メモリで構成されたモジュールをベースにして，最大 128 の RISC プロセッサによる同時並列処理が行われる．この構成を示したのが図 9 である．

XTPA による並列処理は，ノンストップ・システムを基本コンセプトの一つとしているが，中核としての XPC は図 9 に示すように，プロセッサ，メモリ，電源，すべてがフォールト・トレラント構造になっている．また，機能としては次のものがある．

① 高速レコード・ロック制御機能

複数ホストの並列処理におけるシングル・システム・イメージの大規模トランザクション処理を実現するために，複数ホストからのロック要求を高速に制御する．これにより同一トランザクション処理プログラムの複数ホストでの同時走行が可能となる．

② データ入出力処理を高速化するグローバル・キャッシュ機能

ディスクからのデータの読み込みでは，ディスクから読み込まれたデータを

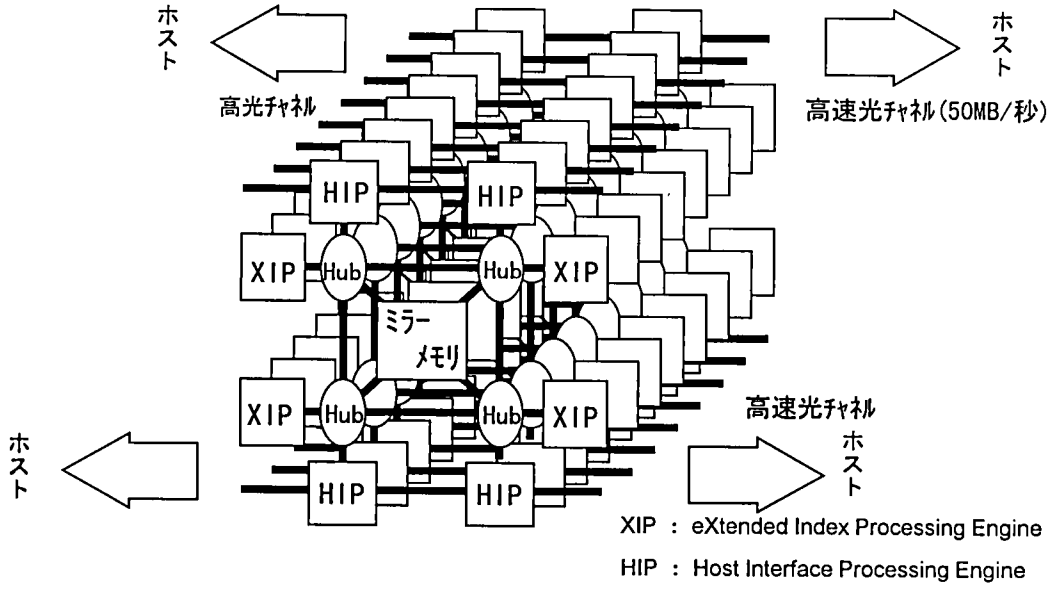


図 9　高度並列処理を実現する XPC の内部構造

XPC上に保持し，その後のデータ・アクセスはXPCのメモリを介して行う．また，書き込みについては，XPCのメモリ上でのデータ更新時点でホスト側に完了を通知する．これにより，ほとんどのデータ・アクセスがXPC上で完了することになり，データの入出力処理が飛躍的に高速化(1ミリ秒)する．グローバル・キャッシング機能はXTPAを構成するすべてのホストに対して有効であり，複数ホストからの特定ディスクに対する入出力の競合にも有効である．

③ レジデント・ファイル機能

XPC内のメモリにファイルを常駐化し，恒常的な高速ファイル・アクセスを保証する機能である．

④ プリフェッチ機能

シークエンシャル・ファイルの場合，データ・アクセス時に次の処理で必要なデータをXPC上に先読みし，アクセス時のヒット率を上げる機能である．

⑤ ホスト間通信機能

あるホストのXTPA環境への参入・離脱，ホスト間でのトランザクション負荷分散，あるいは各ホストのメモリ上のデータ・リフレッシュなどの際にXPCを介してホスト間通信を行う．

また，XPCはキャッシング機能により，トランザクション処理のみならず，全社規模のデータベース処理，バッチ処理，およびデマンド処理などで処理するあらゆるデータ・モデルの入出力を大幅に高速化することができる．図10はXPCを導入したあるユーザの夜間バッチ処理時間が大幅に短縮された例である．

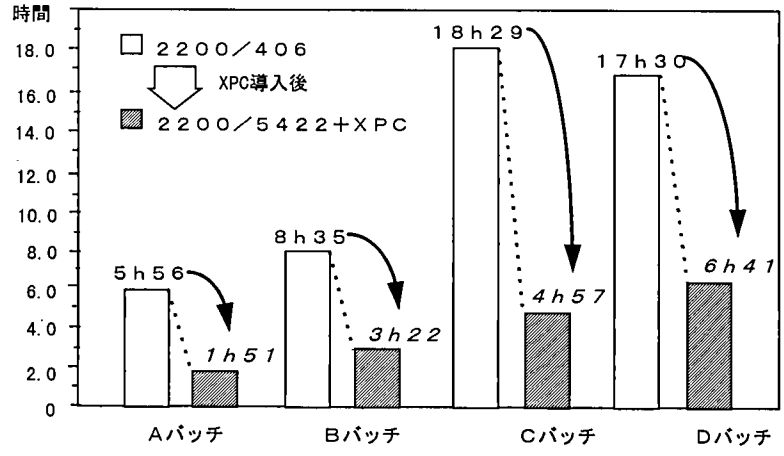


図 10 あるユーザでのXPC適用効果（バッチ処理）

3.4 オープン環境との共存・協調の実現

当社は，新世代オープン・エンタープライズ・サーバITASCAの発表に合わせて，これからの企業情報システムのモデル（エンタープライズ・サーバを中核としたエンタープライズC/Sシステム）を提示した．同時に，エンタープライズC/Sシステムを具現化するための基本的な考え方であるClearPathと，プロダクトとサービスから成るClearPathソリューションセットを提案している．

エンタープライズC/Sシステムの中核をなすサーバとして発表されたITASCA 3800は，当然ながら次に紹介するように，オープン環境との共存・協調を基本コンセプトの一つとしている．

全世界的規模のネットワーク構築と，ネットワーク上の膨大な資産（データ・情報）へのアクセスを考えたとき，ネットワークへの親和性は必須であり，技術的な基盤としてハードウェア面では，イーサネット*4，FDDI，さらにATM（計画）などをI/Oチャネル・アダプタ直結とし，オープンなネットワーク環境への接続性をさらに向上させている．また，ソフトウェア面では，次の対応がなされている．

① 標準OS（UNIX　SVR 4）環境の実現

OS 2200上にUNIX SUR 4環境を提供する．RPC（Remote Processor Call），NFS（Network File System），FTP（File Transfer Protocol）などによりPC，UNIXとの連携基盤を確立することができる．また，OS 2200下で稼働するアプリケーションをUNIX環境で開発，実行することが可能となる

② 標準（ODBC対応）のデータベース・アクセス

標準のインタフェースでITASCA 3800上のRDMS，DMS（計画）へのアクセスを可能とする．UNIX機，NTサーバ上の各種データベースと全く同様なインタフェースでITASCA 3800上のデータベースが利用できる．

③ 分散ファイル・アクセス

TCP/IPのFTPプロトコルに基づいたPC，UNIX機との間の高速ファイル転送と，NFSプロトコルに基づいたエンタープライズ・サーバ上の資源（ファイル）のPCからの共用が可能である．

④ X/OPEN　DTPモデル準拠のトランザクション処理

ITASCAを中核とした2階層，3階層の分散トランザクションが可能であり，2フェーズ・コミット機能も提供する．

⑤ 非同期メッセージによるプラットフォーム間のアプリケーション連携

RPCなどの同期型と異なり，即応性はないが柔軟性のある分散アプリケーション連携ができ，ビジネスの流れに対応したシステム設計が可能である．また，運用が容易であり，モジュール単位に分割した開発が可能などのメリットがある．

⑥ 分散環境での運用支援

— ネットワーク上の部門サーバ，PCに散在するデータをエンタープライズ・サーバ側で自動的に一括バックアップする．

— 複数プラットフォームの運用の集中化，自動化を支援する．

— オープンなプロトコル（TCP/IP）で接続されたエンタープライズ・サーバ，部門サーバ(計画)，およびPC間での分散プリント環境を実現する．

以上のように，オープン環境との共存・協調のポイントである相互運用性，移植性に優れたソフトウェアが提供されるだけでなく，エンタープライズ・サーバとして分散環境全体をとらえた時に必要な運用・開発・実行支援ソフトウェアが用意されている．

4. おわりに

本稿では，C/S モデルを中核のアーキテクチャとして採用した新世代企業情報システムのモデルとそこにおけるメインフレームの位置づけと有効性，さらにそれを実現する要素技術について新世代のメインフレームである ITASCA 3800 で採用された技術を中心に述べた．メインフレームとオープン・システムのそれぞれの良さを生かし，ビジネス環境の変化に素早く，柔軟に対応できる新世代企業情報システムの検討の一助になれば幸いである．

ITASCA に続く次世代メインフレームは，CMOS 技術の更なる進歩によって一層の小型化，低価格化が進み，さらに，ソフトウェアを中心とするオープン化の一層の強化により，オープン環境でのエンタープライズ・サーバとしてその役割を果たしていくものと考える．

最後に，企業情報システム・モデルと ClearPath の検討を筆者等 2 名と共に行った ESS 推進室の東野，林の両氏および多くの助言を頂いた迫畑室長に御礼申し上げたい．

＊1 UNIX は X/Open カンパニーリミテッドがライセンスしている，米国ならびに他の国における登録商標である．
＊2 ClearPath は Unisys 社の登録商標である．
＊3 ODBC は，Microseft 社の商標である．
＊4 イーサネット（Ethernet）は，米国ゼロックス社の商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

参考文献 ［1］ 社団法人 日本電子工業振興協会，「2001 年コンピュータシステム技術に関する調査研究（最終調査報告書）」，1994 年 3 月．
［2］ 特集 クライアント/サーバの難問 開発工数の見積もりに挑む，日経コンピュータ，1995 年 3 月 20 日号．
［3］ 伊東 玄，ユニシスの次世代汎用機，日経ウォッチャー IBM 版別冊「出揃った並列汎用機とディスク・アレイ」，1995 年 11 月．

執筆者紹介 今 江 泰（Yasushi Imae）
1972 年神奈川大学工学部電気工学科卒．同年日本ユニシス（株）入社．ネットワーク・アーキテクチャの調査及び標準化活動への参画の後，ネットワーク・ソフトウェアと分散処理ミドルウェアの商品企画に従事．現在，エンタープライズサーバ企画推進部 ESS 推進室に所属．情報処理学会会員．

丑久保 年常（Toshitsune Ushikubo）
1974 年早稲田大学理工学部電子通信学科卒，同年日本ユニシス（株）入社．U-494，およびシリーズ 2200/1100 のオペレーティング・システムの開発，および保守業務に従事．現在，エンタープライズサーバ企画推進部 ESS 推進室に所属．

シリーズ2200における 大規模トランザクション処理を実現する要素技術

Technologies for High-Volume Transaction Processing on Series 2200 Systems

城川孝二

要約 オペレーティング・システム（OS）の役割は，適切な仮想化を通してシステム資源を抽象概念として使用者に示すと共に各資源を最大限に利用することにある．システム資源の代表的なものには，メモリ，入出力装置およびCPUがある．

メモリについては大容量を実装するシステムが多くなってきており，相対的にはかつて程の重みがなくなってきている．大容量のメモリを如何に利用するかに関心が移っている．

入出力装置自体については，トランザクション処理において重要なのはディスク装置であるが，先取りがその性格上できない（現実的でない）等から，OSが効率面で関与できる余地は余りない．半導体ディスク，キャッシュ・ディスク，XPC等の装置を扱えるようにすることが眼目である．また，入出力処理の負荷がOS全体の負荷に占める割合が大きいことから，その負荷の軽減が重要となる．

CPUは，過去もそして現在も重要な資源であり，そのスケジューリングを司るプロセス・スケジューラ（ディスパッチャ）はOSの核である．シリーズ2200は，代表的なスケジューリング方式であるラウンド・ロビン方式と多段帰還方式を巧みに使い分ける柔軟性を持ち，トランザクションを始めとする多様なアプリケーションを効率的に実行する．また，トランザクション処理（TIP）スケジューラは，順序木構造のメッセージの待ち行列と負荷の小さなスケジューリング方法により，ディスパッチャの柔軟性と併せて，業務要件，応答時間制約あるいは動作特性に合わせた優先度でトランザクションを処理することができる．

Abstract The role of an operating system is to present an abstract representation of system resources to the user via an appropriate level of virtualization while allowing each of these resources to be used to their fullest potential. Typical examples of system resources include the memory, I/O devices and CPU.

Recently, many systems have come to include such a large amount of memory that the importance of memory management has lessened compared to what it was. Most of the attention regarding memory has shifted toward how to utilize such a large amount of memory.

As far as I/O devices themselves are concerned, disk devices have the greatest impact on transaction processing. However, since the physical characteristics of disks do not allow preemptive scheduling (it is not realistic to do so), there is not much room left for the operation system to contribute to disk performance from an efficiency standpoint. Therefore, the main goal of an operating system is to be able to handle a variety of disk devices such as semiconductor disks, cache disks and extended processor complex (XPC) devices. It is also important to ease the overall load on the operating system when large portions of its activity are being taken up by I/O processing.

The CPU exitsts, existed and might exist as an important resource. The process scheduler (dispatcher) is the core of operating system. The 2200 Series systems are so flexible that they can appropriately select round-robin or multilevel-feedback scheduling in order to efficiently run diverse applications

including transactions. In addition, by using a ordered-tree structure and the scheduling system with minimum overhead, the transaction processing (TIP) scheduler in conjunction with the flexible dispatcher can process transactions according to the priorities based on task requirements, response-time constraints, and behavior characteristics.

1. はじめに

多種多様なアプリケーションを無秩序に一台のコンピュータ上で実行することはない．時間帯を分けてあるいは複数のコンピュータを業務形態別に運用するのが普通である．オンライン業務の時間帯にはオンライン・トランザクション処理（OLTP）が，オンライン業務終了後にはオンライン業務で更新されたデータベースのセーブや再構成，マスター更新あるいはレポート処理等のバッチ処理が主要な業務となる．

しかし，時間帯が別ではあるが，トランザクション処理とバッチ処理の両方を処理することに変わりなく，したがって，それぞれを効率良くこなすシステムが必要であることに変わりはない．また，オンライン業務中にもディレード・バッチ等がバックグランド・ジョブとして定期的に稼働するのが普通である．この面では，トランザクション処理とバッチ処理との共存が求められる．

一方，アプリケーション開発（会話型処理）について言えば，本番業務への影響（効率の劣化やデータベース破壊）を避けるため，開発専用機が設置されることが多い．この場合でも，無停止システムでの待機機としての役目を開発専用機が担い，一朝ことが起これば本番機としてオンライン業務を引き継ぐ役目を担っていることが多い．したがって，開発専用機とは言っても，機能的にはトランザクション処理もバッチ処理も可能でなければならない．つまりは，基幹業務を担うシステムには，トランザクション処理，バッチ処理さらには会話型処理のそれぞれを効率良く実行する能力が求められる．このような様々な処理形態に適応できるように設計開発され進化してきたのがメインフレームである．メインフレームは，単に総花的に何でも程々にこなせると言ったものではなく，生まれ育ってきたのにはそれなりの必然性があったのである．

しかし，全てをメインフレームで処理する時代でなくなったことも事実で，GUI 等の使用者インタフェースや気象予測のような膨大な計算を必要とするアプリケーションは，PC/WS（ワークステーション）やスーパコンピュータが担うのが自然である．今日，メインフレームに期待される役割は，様々なコンピュータを繋いだネットワーク上で基幹データベースを中心とする全社レベルの処理系を担うオープン・エンタープライズ・サーバである．最近は，データ蓄積用としても有用であるとの評価も生まれており，CMOS 技術の採用等によるメインフレーム自身のダウンサイジングと相俟ってメインフレーム見直し機運が高まっている．

本稿の目的は，基幹業務システムを担う上でシリーズ 2200 のオペレーティング・システム（OS）である EXEC が採用している様々な技術の一端をトランザクション処理に焦点を当て，構造解析レベルに陥ることなく一般論や UNIX*1 との対比等も交えて，紹介することにある（オープン対応については，当号の別稿で論じている）．まずマルチプログラミング技術を概観し，次にトランザクション処理（TIP）スケジューラおよびプロセス・スケジューラ（ディスパッチャ）の特徴を明らかにする．

2. マルチプログラミング

マルチプログラミングとは，複数のプログラムを同時に処理しているとの錯覚であり，利用者各々が自分用の仮想的なプロセッサ（実際のハードウェア・プロセッサより遅いが）を持っているかの如くのみせかけである．仮想プロセッサの数には，システム資源への過度の競合に伴う負荷の増大やシステム情報の保持領域の枯渇を避けるため，ある一定の上限が存在する（最大許容多重度）．たとえば，シリーズ2200におけるバッチ処理のマックス・オープン，トランザクション処理のTPMAXである．

多重度を制約する要因は，メモリ容量，入出力のバンド幅あるいはCPU能力を始めとするシステム資源量，およびそれらを有効利用するための方式技術である．

2.1 メモリ

メモリは，値段の低下に伴い大容量のメモリを実装するシステムが多くなり，仮想メモリ技術（アドレス空間，実メモリ空間，アドレス・マップ）の進歩もあって，多重度向上の阻害要因となることが少なくなってきている．むしろ，巨大な仮想アドレス空間と大容量の実メモリをどう使いこなすかが課題である．

シリーズ2200は，EXECやデータベース・マネージメント・システム，ソートあるいはコンパイラ・システム等のシステム・ソフトウェアが大容量のメモリを活用し入出力要求を減らしてシステム性能の向上を図っている．と同時に，使用者が従来プログラムを変更することなく大容量のメモリの恩恵を享受できるサイバー・バッチ機能(EXFILE/EXPIPE)も提供している．EXFILEは，メモリをファイルとして利用することでディスク装置への入出力を減らし処理の高速化を図るものである．EXPIPEは，遂次処理を並列化することで処理時間の短縮を図るものである．

2.2 入出力

入出力処理負荷がOS全体に占める割合が多いことから，その負荷軽減が入出力処理における主要課題の一つである．負荷の一部は，ファイルの内部表現，とくに論理アドレスから物理的なディスク領域アドレスへの変換のための構造，が深く係わっている．

UNIX（S5ファイル・システム）では，ディスク領域の割当をブロックと呼ぶ固定長の単位で行う．ファイルを論理的にブロックの配列と見なし，ブロックの物理（ディスク領域）アドレスを示すエントリの配列（論理ブロック番号を添え字とする）を変換に使用する．ファイルが大きいと，配列が大きくなり配列がinodeに納まらず，最悪3段階の間接ブロックが必要となる．この結果，変換に際して間接ブロックのアクセスが余分に必要となり負荷が増大する．テキスト編集等で使用している分にはファイルも小さく間接ブロックが生じたとしても1段階程度であろうが，データ規模が桁違いに大きいOLTPでは効率の面で甚だ不都合である．

一方，シリーズ2200はどうかと言うと，UNIXと類似の構造を(したがって，同じ様な問題も）持っていた時代があった（GTBあるいはグラニュール・テーブルがブロック・エントリの配列に相当)．ディスク領域の割当をトラックと呼ぶ1792語（36ビット/語)の固定長の単位で行うことは変わらないが，今日では，できるだけ連続した領域を割り当てるようにし，その連続領域の物理アドレスや大きさを一つのDAD

(Device Area Descriptor) エントリに記述するように変わっている．論理的には，ディスク装置一台分の大きさのファイルを一つのエントリで記述することも可能である．最大8エントリを一つのDADテーブルに納めることができる．

OLTPにおいては，ファイルの大きさの見積りはかなりの精度で可能であり（また必要でもある），ファイルに必要な分の領域を事前に確保して置くのが普通である．また，ファイルの作成や削除を頻々と行う，と言ったこともない．したがって，領域の空きが簾状になることもなく，大きな連続領域の割当に窮することもない．ファイルが大きくても変換用のエントリが増えないので，変換に伴う負荷の増加の恐れもない．ファイルの拡張が必要となったとしても(任意の大きさに拡張可能)，拡張分をファイルの最後尾の領域からの連続領域に割り当てようと（エントリが増えない様に）するので，無闇なエントリの増加をもたらすこともない．実際，どんなにファイルが大きくてもDADテーブルが複数となる（エントリが8を超える）ことはほとんどない．

図1に，大きさがそれぞれs1とs2の二つの連続領域が割り当てられているファイルを例にとりUNIXとの表現方法の相違を示す．UNIXでは，ブロック長が1Kバイトとすると，64Mバイトを超える場所のデータをアクセスする場合は3回，256Kバイト超の場合は2回，10Kバイト超の場合は1回の間接ブロックの参照が必要なのに対して，シリーズ2200ではそのような余分な参照は不要である．

トランザクション処理において重要なのはディスク装置であるが，先取りがその性格上できない(現実的でない)等から，OSが効率面で関与できる余地は余りない．入

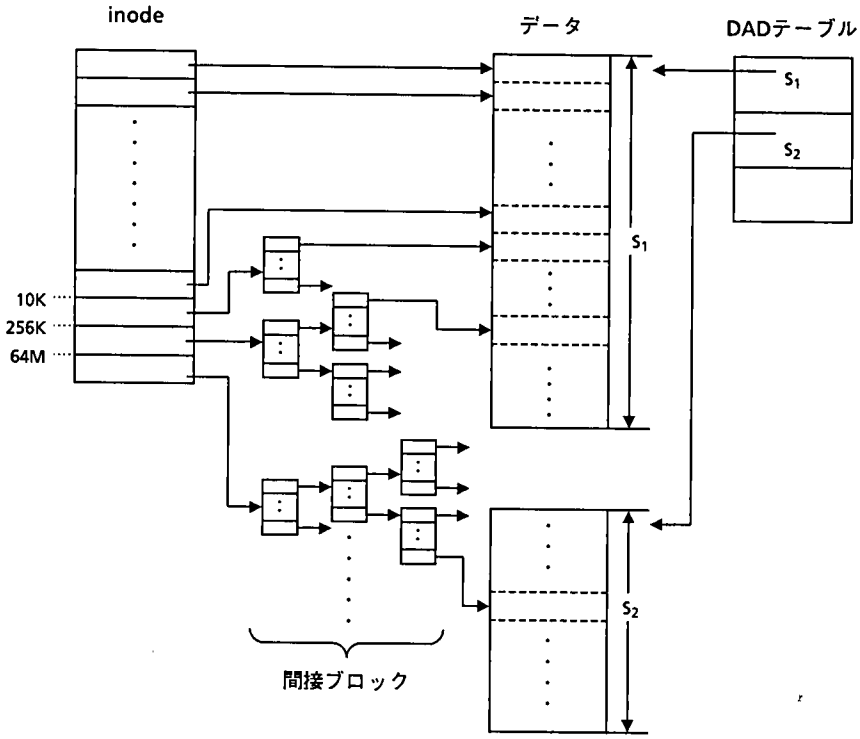


図1　UNIXとシリーズ2200とのファイル領域表現方法の相違

出力に係わる理論としては，ディスク・ヘッドの移動距離を短くし，アクセス時間を短くする（しかし，サービスに片寄りが生じる）SSTF (Shortest-seek-time－first)/SATF (Shortest-access-time－first) 方式と公平なサービスを図る FIFO 方式との間の折衷方式である SCAN 方式がある程度である．しかし，入出力処理時間が経過時間全体に占める割合は大きく，また，格納データの量も増加の一途であり，その重要さはいや増している．そのため，データ転送能力が飛躍的に向上する半導体ディスク，キャッシュ・ディスクあるいは拡張データ処理装置（XPC: eXtended Processor Complex）が登場し普及している．それに伴いそれらを扱うドライバ（あるいはハンドラ）の整備が必要となるが，そのためには，新たなドライバの開発・導入が容易な，たとえば，使用者プログラムとして実現できるような拡張性や柔軟性に富んだ構成の OS が必要である．

2.3 CPU

SMP（対称型マルチプロセッサ機）は 16 台の CPU が限界と言われており，実体としての CPU を仮想プロセッサに割り当てる方式が CPU の利用率のみならずメモリや入出力装置の効率的使用にも大きな影響を及ぼす．シリーズ 2200 は，登場の当初からマルチプロセッサを前提としたアーキテクチャであり，多年の実績がある．

一般的には，ジョブ（処理要求）のほうが仮想プロセッサの数（最大許容多重度）より多く，仮想プロセッサの数の方が CPU の数より多い．したがって，図 2 のように，ジョブに仮想プロセッサを割り当てる機能と仮想プロセッサに CPU の様な物理資源を割り当てる機能とに，スケジューリング機能を分けるのが普通である．

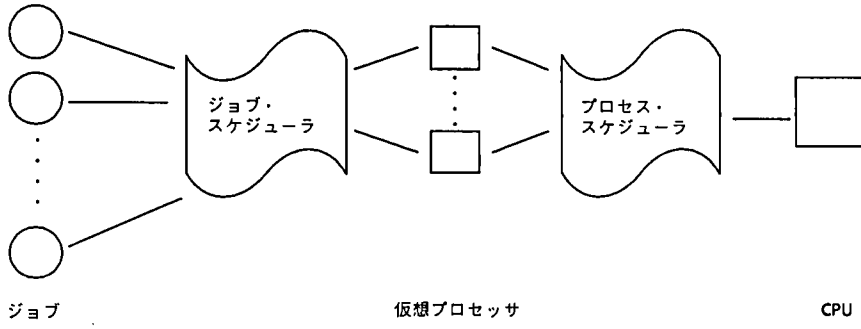


図 2　スケジューリング機能の分割

シリーズ 2200 では，バッチ・ラン（ジョブ）やデマンド（会話型）ランへの仮想プロセッサの割当を制御するスケジューラをコース (Coarse) スケジューラ，トランザクション・プログラムへの仮想プロセッサの割当を制御するスケジューラをトランザクション処理 (TIP) スケジューラと呼ぶ．CPU の割当を司る下位レベルのスケジューラをディスパッチャと呼ぶ．それぞれのレベルのスケジューラは，一般には色々な名前で呼び習わされている．上位レベルのスケジューラをジョブ・マネージャあるいはジョブ・スケジューラ，下位レベルのスケジューラをプロセス・マネージャ，プロセス・スケジューラ（あるいは単にスケジューラ）あるいはディスパッチャと呼ぶのはその例である．あるいは，時間的な観点から，長期的スケジューラ，中期的スケジ

ューラ（プログラムのスワップ制御）そして短期的スケジューラ，との呼び方もある．本稿では，一般論では，ジョブ・スケジューラとプロセス・スケジューラを用い，シリーズ 2200 を論じる場合はシリーズ 2200 の用語を用いる．ジョブ・スケジューラは，プロセス・スケジューラと相俟って，応答性能の良さと言った使用者への便宜供与を図り，プロセス・スケジューラはシステム資源の利用率の向上を図る．

3. ジョブ・スケジューラ

3.1 バッチ/会話型処理のジョブ・スケジューラ

仮想プロセッサに空きがあれば，空いている仮想プロセッサをジョブに割り当て，余裕がなければ，仮想プロセッサが空くのを待つ．仮想プロセッサを割り当てるジョブの選択は，ジョブの形態(バッチ処理，会話型処理)，必要とする資源量の多少あるいは使用者が指定する規範（優先度，所要時間の見積，デッドライン・タイム）等に基づいて行う．会話型処理に関しては，素早い応答が前提の処理形態なので，仮想プロセッサの数に制限を設けず，投入ジョブに直ちに仮想プロセッサを割り当てるのが基本である．シリーズ 2200 においても，システム資源の枯渇等の異常事態発生時を除いて，直ちに仮想プロセッサを割り当てて処理を開始する．

バッチ処理形態のジョブについては，使用者が指定する実行順，デッドライン・タイム，開始時間，見積所要時間，優先度や使用するファイル（ディスク，テープ・ファイル）の状況等を勘案してスケジュールする．優先順位の判断がつかなければ，最も単純な方式である，来た順番にサービスする FCFS でスケジュールするのが普通である．

見積所要時間は，デッドライン・タイムとの関係で，デッドライン・ジョブの優先度にする時点の決定に使用したり，見積処理時間の短い順にサービスする SEPT (Shortest-expected-processing-time) 方式のスケジューリングに利用したりする．SEPT は，処理時間が短い順にサービスする SJF (Shortest-job－first) が事前に正確な処理時間が判っている必要があり非現実的なのに対して，正確性には欠けるが見積時間で処理時間を代替する現実的な方式である．

処理時間の短いジョブに優先権を与える方式は，処理時間の長いジョブの犠牲（大きな遅れ）を伴うが，平均経過時間（ターンアランド・タイム）を短くする上で有効である．平均経過時間あるいは応答時間が短くなれば，システム内に存在する平均ジョブ数が少なくなり各種システム資源（メモリ，ロック，等）に対する競合が減る．競合が減ればそれに伴う負荷が減る．

また，多くのメインフレームでは，ジョブをグループ（あるいはクラス）に分けてそれぞれのグループ内で独自のスケジュールを行うことができる機能も持っている．グループそれぞれに（グローバルな多重度とは別に）多重度や磁気テープ装置等の資源の枠を決め独自の規範を適用する．シリーズ 2200 では，EXEC の機能であるコース・スケジューラがデッドラインや優先度等の基本機能を司り，その上に位置するバッチ運用管理システム（IOF/WORK）が単一ホスト・システムから複数ホスト・システムにわたる幅広い環境下でのバッチ処理業務の統合運用管理機能を提供する (IOF/WORK については「技報　Vol. 14 No. 2」に詳しい)．

3.2 TIP スケジューラ

3.2.1 TIP スケジューラの概要

トランザクション・プログラムの構成や制御構造は，バッチやデマンド・ランが実行するプログラムと基本的に同じである．固有のアドレス空間を持ち相互干渉の恐れがないのも，逆に共通のアドレス空間を持つことでルーチンの共用を図ったりデータの授受を行ったりできるのも同様である．大きな違いは，その性格上，負荷の小さなスケジューラとそれを可能とする制御構造の存在である（図3）．

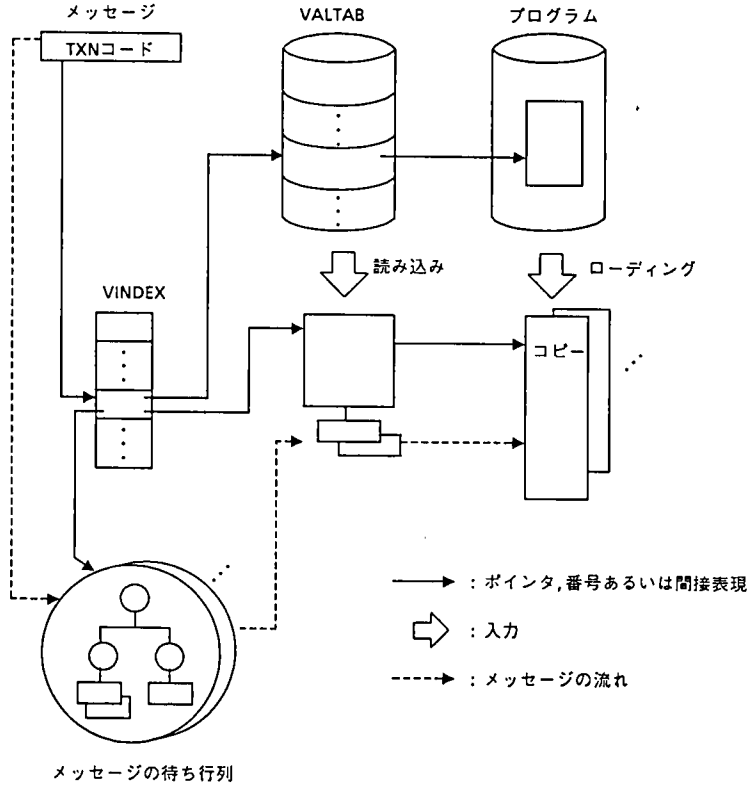


図 3　TIP スケジューラの制御構造

バッチやデマンド・ランの場合はランの属性情報やプログラムの所在場所（エレメント名）等は@RUN と@XQT（あるいは@processor）制御文で指示するが，トランザクション・プログラムの場合は VTBUTL プログラムで事前に VINDEX（Validation INDEX）および VALTAB（VALidation TABle）に登録しておく．VINDEX には，トランザクション・コード，プログラム番号（VALTAB エントリ番号），アプリケーション・グループ番号，入力メッセージの優先度や回復オプション等が入り，VALTAB には，プログラムの所在場所とローディング情報，実行時オプションやマックス・コピー等の各種属性情報が入る．SUPUR プログラムにより，一回の入出力要求で（高速に）ロードできる形式でプログラムをファイルに格納することができる．プログラムを世代に分けてそれぞれを別のファイルに格納すること（プログラムの世代管理）もできる．

メッセージは，ルーチン呼出におけるルーチン名に相当する部分（トランザクション・コード）とルーチンへの引き数に相当する部分からなる．メッセージが入力されると，トランザクション・コードが示すVINDEXエントリ中のアプリケーション・グループ番号とメッセージの優先度を基にメッセージを入力メッセージの待ち行列に入れる（アプリケーション・グループと待ち行列については，後述する）．待ち行列からメッセージを取り上げると，VALTABエントリの情報を基にプログラムをスケジュールする．入力メッセージの多いプログラムについては，ディスク装置上のプログラムのメモリへのコピーを複数作成（ローディング）し並列的に処理する．各コピーは，元のプログラムは同じでも，それぞれが独立したプログラムとして実行する．VALTABエントリをメモリに読み込み展開した構造体には，コピー数と各コピーの制御構造のリストおよびメッセージの待ち行列等が存在する．

プログラムのスケジュールには，メッセージを処理するプログラムをディスク装置からメモリに読み込む（初期ロード），処理すべきメッセージの到着を待っている（スティッキング）プログラムを実行する（リアクト），実行中のプログラムが現メッセージの処理終了後に引き続いて処理する（リスタート），がある．負荷の大きい初期ロードはなるべく避けるようにする．初期ロード以外は，ほとんどスケジューリング負荷が掛からない．また，プログラムをメモリに常駐化（仮想プロセッサを保持）する機能により初期ロードを回避することも可能である．任意のプログラムが常駐化の対象となり得，適宜常駐化したり非常駐化したりできる．常駐化機能は，メッセージを読み込んでは処理することを繰り返す（マルチイニタル）機能と併用することでさらに負荷の軽減を図ることができる．プログラムの初期化と終了処理が初期ロード時の一回だけで個々のメッセージ処理では不要なので，スケジューリング負荷がほとんどない上にプログラムの処理時間の削減ができるからである．なお，マルチイニタルは，非常駐化プログラムでも有効である（リスタートの時に機能する）．実行頻度の高いプログラムを常駐化すれば効率上の効果もそれだけ大きくなる．また，常駐化しなくても，実行頻度の高さ故，プログラムのロードを伴わないリスタートあるいはリアクトの割合が多くなり，負荷が小さく済む．

プログラムは，それぞれのアプリケーション・グループ(APG)内で実行する．APGとは，プログラムやデータベース等をグループ化したアプリケーション固有の運用回復環境である．それぞれのグループは独立した存在であり，他のグループとは独立に，運用管理・回復処理が可能である．本番用と開発テスト用にAPGを分けて，開発テストで起こり得るプログラム・エラーやデータベース破壊等の影響が本番業務に及ばないようにシステムを構築することが可能となる．また，多様なアプリケーションで構成されるシステム，たとえば勘定系を始めとする情報系・国際系・対外系・証券系等のアプリケーションで構成される銀行システムをそれぞれのアプリケーションに分割し運用することも可能である．XTPA (eXtended Transaction Proseccing Architecture) に基づく複数ホスト・システムの場合は，構成ホスト全体を一つのシステムと見なす（ホスト横断の）コンカレントAPGを設けることも，特定のホスト内だけのローカルAPGを設けることも可能である．

3.2.2 入出力メッセージの待ち行列

入力メッセージの待ち行列は，図4のような優先度で順序付けられた順序木である．木はAPGごとに存在し，システム生成時に構造を規定する．葉（あるいは端点）には優先度，最大許容多重度（図ではmaxと表記）および待ち行列が，親には名前，最大許容多重度および走査方法が対応する．メッセージは，対応する優先度の葉の待ち行列に入る．木を根（ルート）から順に節点（ノード）を走査し，最大許容多重度に達していない葉からメッセージを取り上げ，プログラムをスケジュールする．

多重度は，メッセージを取り上げた時に上がり，スケジュールされたプログラムが処理を終えた時（正確には，順序木は統合回復機能と密接に係わっており，プログラムがコミットした時）に下がる．親の最大許容多重度は子の多重度を制限する（子の多重度の和は親の最大許容多重度を超えることはできない）．

図4において，優先度3，4および5の現在の多重度がそれぞれ1，2，1であるとすると，それらの和は親であるRESERVの最大許容多重度の4に達しているので，優先度5は最大許容多重度が2であり多重度に余裕があるがメッセージをそこから取り上げることはしない．根の最大許容多重度はAPGの最大許容多重度を示す．最大許容多重度は，動的に変更することができる．

走査には，優先度順に行う方法と最後に走査された葉の次の葉（隣の葉あるいは先頭の葉）から走査を始める循環的な方法とがある．親が循環方法を示していれば，その子（葉）らの間には優先関係は存在せず，優先度はメッセージを入れる葉を見つける

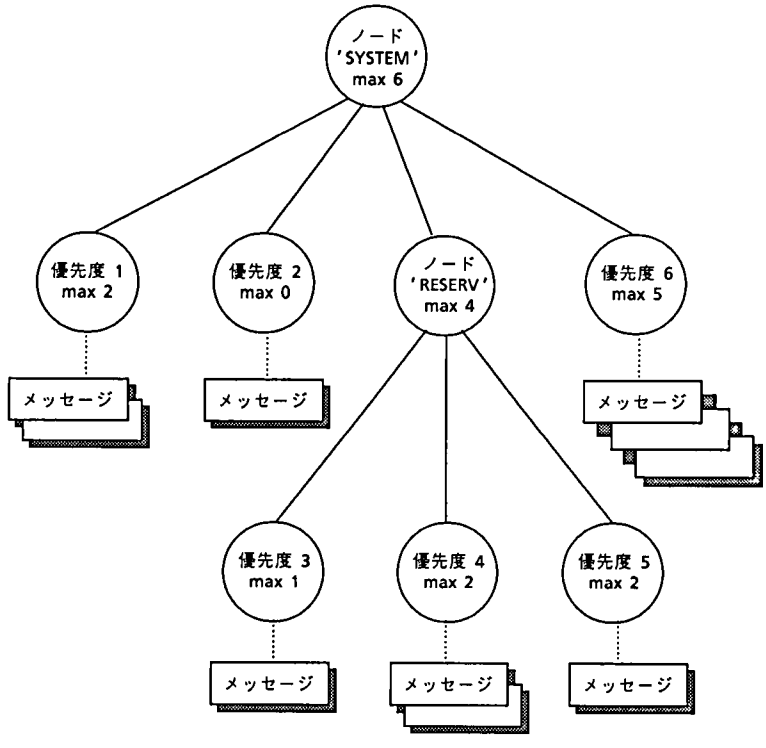


図4　スケジューリング順位木の例

ための単なる標識の意味合いしか持たない．図で根のSYSTEMには優先度順，RESERVには循環走査方法が設定されているとすると，優先度1，2の順で，次にRESERVの子である優先度3，4，5の葉を循環的に，続いて優先度6を走査する．優先度4からメッセージを取り上げた結果，RESERVの子の多重度の和が最大許容多重度の4に達した場合には，RESERVの子の走査はそこで終わる．次のRESERVの走査は優先度4の次の優先度5から始まり，3，4と続く．

APGごとに順序木を設定できるので，アプリケーション間でシステム資源の適正な配分を図ること等ができる．重要度の高いあるいは応答時間制約の厳しいアプリケーションについては最大許容多重度を大きくし処理の促進を図る，あるいは開発テスト用APGの最大許容多重度を小さくし本番業務への効率上の影響を押さえる等である．メッセージにも優先度が設定できるので，会社外の不特定多数（所謂，顧客）のトランザクションの処理は会社内で発生するトランザクションの処理より優先度を高くしたい，等の要求に応えることができる．

それぞれの処理についてもさらに優先度を設ける必要が生じることもある．たとえば，電話での質問に応える音声照会通知システムのトランザクション処理を最優先したい，とかである．このような場合でも，優先度ごとに最大許容多重度を設定できるので，優先度の低いメッセージが何時までも取り上げられない事態を避けることができる．優先度の高い方が最大許容多重度に達すると，その優先度のメッセージを取り上げることはせずより低い優先度のものを取り上げるからである．

4. プロセス・スケジューラ

CPUは最重要な資源の一つである（CPUを必要としない処理は存在しない）．マルチプログラミングは，CPUを割り当てるプロセスをなにがしかの方式でもって順を切り替えること（コンテクスト・スイッチ）により実現する．CPUを割り当てる基本単位を，UNIXではプロセス，シリーズ2200ではアクティビティと呼ぶ．つまり，プロセス・スケジューラ（あるいはディスパッチャ）の機能とは，次にCPUを割り当てるプロセス（あるいはアクティビティ）を選択することである．

近年は，Mach[*2]に見られるように，プロセスの概念を仮想アドレス空間やファイル・アクセス等の実行環境を規定するタスクとそのような環境下で実行するスレッドとの二つに分割する傾向にある．この場合は，スレッドがCPUの割当単位で，タスク内で複数生成することができる．プロセスの起動は，一つのタスクと一つのスレッドの起動に対応する．タスクが分離されている分身軽なところから，スレッドをライト・ウェイト・プロセスとも呼ぶ．まさに，シリーズ2200の出生当初からの概念であるアクティビティに相当する．いくつもの要求を同時に処理するサーバ・プロセスを効率的に実現する上で有効である．要求ごとにスレッドを作成し処理すれば，その内のいくつかがシステム・コールを出して待ち状態になったとしても残りのスレッドは走行可能である．また，マルチプロセッサの有効利用にもつながる．OPUSが採用しているマイクロカーネル（Chorus Kernel）はスレッドを実現している例である．

システム資源の効率的な利用を図るための手法の一つに，入出力要求を頻繁に行う(I/Oバウンド)プロセスに優先的にCPUを割り当てる方式がある．I/Oバウンドのプ

ロセスはCPU制御を受け取ってもすぐに入出力待ちとなり制御を放棄するので，ディスク装置等の入出力機器の動作中に次の優先度のプロセスの実行（入出力処理とCPU処理のオーバーラップ）が可能である．入出力機器をなるべく動作させておくことにより，システムにボトルネックが生じる可能性を軽減し，システム・スループットの向上を図ることができる．I/Oバウンドのプロセスに優先権を与えることによる効率の向上は，CPUバウンドのプロセスの犠牲（ターンアラウンド・タイム/応答性能の劣化）の上に成り立っている．極端な場合は，CPUバウンドのプロセスのスケジュールがどんどん後回しにされて走行することができない状況も有り得る．このようなプロセスの救済を図ることも必要である．

一般に，プロセス・スケジューラは各プロセスに優先度を付け，その優先度に基づいてCPUを割り当てるプロセスを決める．プロセスの優先度は，処理形態により（ジョブ・スケジューラとの関係で）決まる要素，業務の優先度等に基づいて人為的に付与される要素，あるいは入出力処理の割合が大きいとか端末からの入力待ちの割合が大きいとかのプロセスの振る舞い（特性）に基づいてプロセス・スケジューラ自身が動的に付与する要素，等との組み合わせで決まる．I/Oバウンドのプロセスや処理時間そのものが非常に短い等の特徴を持つプロセスに優先権を与えるようにする方式は，入出力処理との多重度を高めCPUの利用率を高める上で有効であり，会話型処理やトランザクション処理での平均応答時間を短くする上で有効である．応答時間が利用者の生産性（単位時間当たりの入力コマンド数）に与える影響は大きく，1秒（Magic one second mark）を切ると生産性が劇的に向上することが知られている．

本章では，先ず代表的なスケジューリング方式とその特徴を述べる．次にUNIXが採用しているスケジューリング方式とその問題点を述べ，最後にシリーズ2200が採用しているスケジューリング方式の特長を述べる．

4.1 代表的スケジューリング方式

スケジューリング方式としては，色々な方式が提案されているが，代表的なものにラウンド・ロビン方式と多段帰還方式がある．

1) ラウンド・ロビン方式（図5）

初期のタイム・シェアリング・システムでよく用いられた方式である．プロセスにタイム・スライス（あるいはカンタム）と呼ぶ一定の時間を割り当て，その時間内に入出力処理待ちになるとか処理の終了とかでCPUを明け渡たさなければ，プロセスの実行を中断し実行可能待ち行列の最後尾に入れる．次にサービス

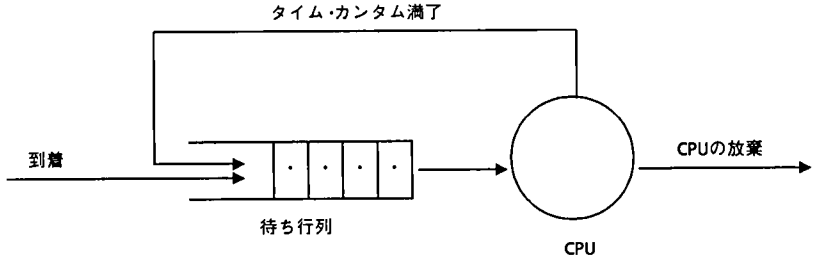


図5　ラウンド・ロビン方式

を受けるのは，実行可能待ち行列の先頭のプロセスである．特定のプロセスがCPUを長時間占有することはない．処理時間が短ければ待ち時間も短い(応答時間が短い）ので，会話型処理に適している．仕組みが単純なのでスケジューリング負荷が小さい．残り処理時間の短いプロセスほど早く終了するので，残りの処理時間が短いプロセスを優先的にサービスするSRPT（Shortest-remaining-processing-time）方式の一形態と見なすことができる．タイム・カンタムの大きさは，効率に影響を与えること大がである．大きすぎるとFCFS方式に近づき，小さ過ぎるとタイム・カンタムの使い尽くしに伴う（割り込み処理やコンテクスト・スイッチの）負荷が大きくなる．これが最適値であると言ったものはなく，マシーン・スピードやアーキテクチャ，さらには使用目的等を勘案し，経験的に求めるのが一般的である．

2) 多段帰還方式（図6）

ラウンド・ロビン方式の変形であり，多くのシステムが会話型処理等で採用している．プロセスの実行に連れ優先度がある範囲で変化する（たとえば，小さい数字が高い優先度を意味するとすると1〜n)．優先度に待ち行列が対応する(待ち行列を制御する構造体を要素とし優先度を添字とする一次元の配列)．

プロセスの開始時は最高の優先度（たとえば，1）を割り振り，その優先度の待ち行列の最後尾に入れる．プロセスには，ラウンド・ロビン方式と同様に，タイム・カンタムを割り当てる．タイム・カンタムを使い尽くすと（その間にCPUを放棄しなければ)，優先度(p)を一段下げた優先度(p+1)の待ち行列の最後尾に入れる．次にCPUサービスを受けるのは，その時点での最高の優先度の待ち行列の先頭のプロセスである（直前に優先度をp+1に落とされたプロセスである可能性もある)．端末からの入力あるいは入出力の完了等でプロセスが待ちの状態から実行可能な状態になったときには優先度を高くする（たとえば，1にする)．そ

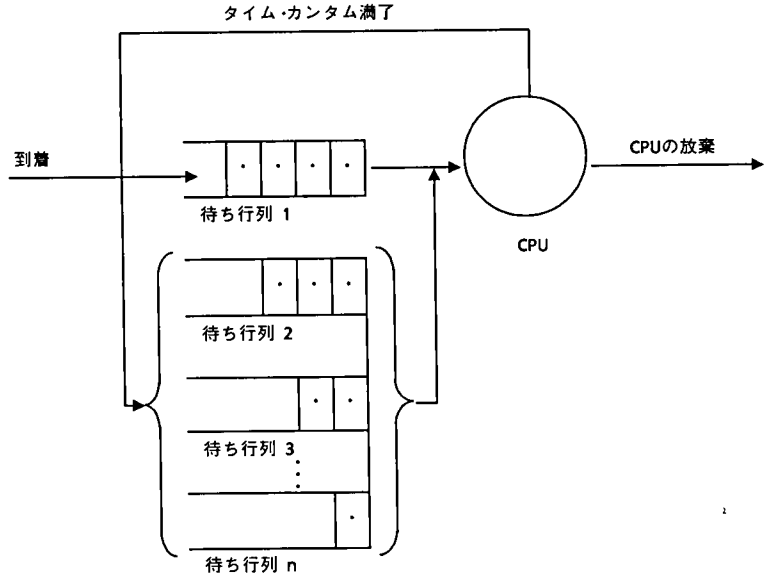


図 6　多段帰還方式

の優先度が実行中であったプロセスの優先度より高い場合には，CPU を先取りする．実行中であったプロセスは，その優先度の待ち行列の先頭に入れる．処理時間が長いあるいはCPUバウンドのプロセスは，徐々に優先度が落ちて行くことになるが，最低の優先度(n)にまで下がると，そこでラウンド・ロビン方式のスケジューリングを行う（タイム・カンタムを使い尽くすと，同じ優先度の待ち行列の最後尾に入れる）．

タイム・カンタムは，優先度の関数“TQ(p)＝q＊2＊＊(p−1)”で与えてタイム・カンタムの使い尽くしに伴う（割り込み処理やコンテキスト・スイッチの）負荷の軽減を図る方法もある(シリーズ2200で類似の方法を採用)．q は，ラウンド・ロビン方式でのタイム・カンタムと同様に，経験的に求める．この方法の起源は非常に古く，1962年に提案されCTSSシステムで採用された（その当時は，q はプログラムのスワップに要する時間）．

この方式は，処理時間の短いプロセスやI/Oバウンドのプロセスを優先的にサービスするので，ラウンド・ロビン方式よりも一層SRPT方式に近く，システム・スループットの向上が望める．とくに，処理時間の短い（あるいはI/Oバウンドの）プロセスと処理時間の長いCPUバウンドのプロセスが混在する（CPUサービス時間の分布の変動係数が大きい）場合に効果的である．しかし，緊急性とは無関係にスケジュールするので，応答時間の保証が必要なOLTPには適さない．また，CPUバウンドのプロセスにはなかなかCPUが割り当てられないことになり，公平さの面で問題が生じる．実際のシステムに於いては，定期的に待ち行列を監視してサービスを長時間受けていないプロセスの優先度を上げる，等の救済処置を施すのが一般的である．

4.2 UNIXのプロセス・スケジューラ

UNIXには，多くの変種が存在するが，ここでは大きくSYSTEM V系とBSD系とについて簡単に触れる．共に会話型処理（時分割システム）向きのスケジューリング方式であり，カーネル・モードとユーザ・モードの大きく二つに優先度を分ける．カーネル・モードでは待つ事象により（たとえば，ディスク装置への入出力完了）一意に優先度が決まる．ユーザ・モードについては共に多段帰還方式を採用しているが，優先度の求め方に相違がある．

1) SYSTEM V系

プロセスの優先度は，実行可能待ち行列に入っているプロセスについて，1秒ごとに次の関数を施して計算するCPUの使用状況に基づいて決める．

$$DECAY(CPU)=CPU/2$$

優先度は，このCPU使用状況を変数とする次の式で求める．小さな数字が高い優先度を示す．なお，ユーザ優先度ベースはカーネルとユーザの優先度との境界であり，niceは使用者が設定可能な値(標準値は0)である．負の値であれば優先度が高く（スーパ・ユーザのみが可能），正の値であれば低くなる．ある程度は，使用者が優先度制御に係わることが可能である．

$$優先度=CPU使用状況/2+ユーザ優先度ベース+nice$$

この方式は，テキスト編集のようなCPUを使って処理している時間より端末

からの入力を待っている時間の方が長いような会話型処理のプロセスが良くサービスされる（したがって，応答性能に優れている）方式である．CPUバウンドのプロセスについても，サービスを受けない時間が長くなってくると徐々にその優先度が高くなるので，何時までもサービスされない最悪の事態は避けることができる．

2) BSD系

プロセスの優先度がCPUの使用状況に基づいて決まることは，SYSTEM V系と同じであるが，DECAY関数および優先度を求める計算式が異なる．なお，loadは過去1分間の平均実行可能待ち行列の長さを示す．

$$\mathrm{DECAY(CPU)} = (2 * \mathrm{load}/(2 * \mathrm{load}+1)) * \mathrm{CPU}+\mathrm{nice}$$

$$\text{優先度} = \text{CPU使用状況}/4+\text{ユーザの優先度のベース}+\mathrm{nice}$$

以上のスケジューリング方式は，会話型処理には向いていても即時応答を要するOLTPには向いていない．会話型処理でも素早く応答することが重要であるが緊急性までは必要としていない(数秒余分に待つことは可能)．応答時間がその時その時のシステム状況に大きく左右される（応答時間の保証ができない）からである．会話型処理用とは別に，緊急性等で優先度を決めることのできるOLTP向きのスケジューリング方式が必要である．

このため，優先度クラスと呼ぶ概念が導入され，会話型処理向きのスケジューリング方式に加え，OLTP向きの方式も提供されるようになった．優先度クラスにはシステム・クラス（カーネル専用），時分割クラスおよびリアルタイム・クラスがあり，リアルタイム・クラスのプロセスは時分割クラスのプロセスより優先度が高い．時分割クラスのプロセスは前記の方式で優先度が決まる．リアルタイム・クラスには複数の優先度が存在し，先取りありのスケジューリングを行う．リアルタイム・クラスのプロセスの優先度は，時分割クラスのプロセスとは異なり，動作特性によらず固定である．リアルタイム・クラスのプロセスは，高い優先度を定常的に保持するので，即時応答を要する業務に適しているが，一方では，プロセスがループしCPUを占有してしまう恐れがある．一旦そうなると，プロセスを終了させる術がなく，再度ブートする他ない．現状では，これを恐れてリアルタイム・クラスの使用を制限している場合が多い．

4.3 シリーズ2200のプロセス・スケジューラ（ディスパッチャ）

4.3.1 優先度クラス

シリーズ2200は，処理形態を高位EXEC，リアルタイム，低位EXEC，トランザクション，デッドライン・バッチおよびバッチ/デマンドに分けてそれぞれにスケジューリング方式（優先度クラス）を設けることができるのが大きな特長で，各種アプリケーションの重要度や特性に応じたきめ細かな制御が可能である．

スケジューリング方式と優先順はシステム生成時に指定することができ，処理形態をさらに分けて別々のスケジューリング方式を適用する（クラス分けする）ことでさらに多様なクラス構成とすることも可能である．一般的には，図7に示す標準優先度クラス構成で十分である(上が高く下が低い優先度)．トランザクション処理に対して4種類(内一種はバッチやデマンド処理と共用)のクラスを提供しているので，入力メ

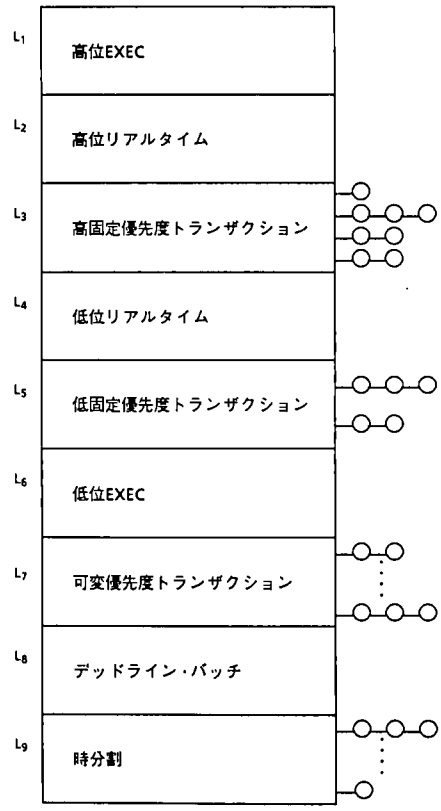


図7　標準優先度クラス構成

ッセージの優先度と併せ，緊急度やアプリケーション特性に合わせた柔軟な対応が可能である．なお，串団子のような図形は，それぞれの優先度の実行可能アクティビティの待ち行列を示している．

各クラスには複数の優先度が対応し，数はクラスにより異なる．たとえば，(L8—L7) は可変優先度トランザクション・クラスの優先度の数である．優先度は，クラスとクラス内での相対優先度により決まる整数値である(シリーズ2200では，処理形態とスケジューリング方式が一対一で対応していた過去の歴史的な経緯から，クラスをタイプ，相対優先度をレベル，とも呼ぶ)．L1 (=1) が最高の優先度で値が大きいほど低い優先度を示す(後述するように，優先度が0の特殊な実行形態が存在する)．優先度はアクティビティの属性であり，実行中のアクティビティの優先度より高い優先度のアクティビティが実行可能となると，優先度の高いアクティビティがCPUを先取りする．

アクティビティがどの優先度で実行するかは，処理形態により異なる．

1)　トランザクション処理

優先度はVTBUTLプログラムで設定（あるいは変更）するプログラム属性の一つである．優先度は，インデックスで間接的に指定する．インデックスは，優先度クラスと相対優先度の対（したがって，優先度）に対応する．指定できる優先度クラスは次の何れかである．

・高固定優先度トランザクション

・低固定優先度トランザクション
・可変優先度トランザクション
・時分割

初めの二つのクラスのスケジューリング方式は，優先度が固定のラウンド・ロビン方式である．後の二つは，優先度が変化する多段帰還方式である．

2) デッドライン・バッチ処理

デッドライン・バッチ優先度クラスに属する．スケジューリング方式は，多段帰還方式である．

3) バッチ/デマンド（会話型）処理

時分割優先度クラスに属する．スケジューリング方式は，多段帰還方式である．なお，トランザクション処理プログラムもこのクラスで実行することが可能である（VTBUTL でそのように指定すれば）．オンライン・バッチ処理は，トランザクション・メッセージにより起動される点ではトランザクション処理の一種であるが，処理内容は長時間走行するバッチ処理そのものであるので，ディスパッチャはバッチ処理として扱う．

アクティビティをリアルタイム優先度で実行する（あるいは元の優先度に戻る）ための EXEC インタフェースが存在する．使用者は，高位と下位リアルタイム優先度クラスを通しての相対優先度を指定する．相対優先度が高位の優先度数(L 3—L 2)以内であれば高位の，超えていれば低位のリアルタイム優先度で実行する．要求したアクティビティのみがリアルタイム優先度となるので，リアルタイム優先度のアクティビティと時分割優先度のアクティビティが混在するプログラムも存在し得る．

なお，リアルタイム優先度クラスには，EXEC 優先度クラスと同様に，タイム・カンタムの概念はない（非常に大きなタイム・カンタムを与えるので，実質的には，そのような概念はないのと同じ)．したがって，リアルタイム優先度の使用には注意が必要である(ループしても強制的に終了させることは可能)．たとえば，通信制御プログラム等のように，一つの電文が到着するや否や即処理し次の電文の到着を待つ（その間 CPU を解放する）とかで緊急性を要する処理ではあるが処理に要する時間は短くて待っている時間の方が長い，したがって，CPU を使う割合が小さいプログラム等で使用する特殊な優先度クラスである．また，時間的制約のある周期的なバッチ処理，たとえばディレード・バッチ，に対してもリアルタイム優先度は有効である．この場合は，低位リアルタイムとすることで，高位リアルタイムで実行する通信制御プログラムの電文処理や緊急性の高い（高固定優先度の）トランザクション処理を妨げないようにすることができる．

4.3.2 トランザクション処理の優先度クラス

高固定優先度トランザクション・クラスより高い優先度のクラスは，高位 EXEC と高位リアルタイムだけである．高位 EXEC は，CPU，メモリや入出力装置の障害通知・回復処理やプログラムの強制あるいは異常終了処理用なのでほとんど使用されることはない．リアルタイム優先度についても先に述べたようにこれまた CPU の使用率が低い．また，低位 EXEC より優先度が高いので低い優先度の（たとえば，バッチ）処理が間接的に実行を妨げることもない．したがって，処理が滞る恐れはほとんどなく，

固定優先度のラウンド・ロビン方式であることもあり，応答時間の予測・保証が可能である．低固定優先度トランザクションについても，低位リアルタイムより低いだけで高固定優先度トランザクションと基本的に同じである．可変優先度トランザクションは，優先度が可変であることおよびバッチ処理等の低い優先度の影響を間接的に被ることになるので先の二つの固定優先度トランザクションのようには応答時間保証に適していないが，スループットの向上が期待できる．

低い優先度の処理がより高い優先度の処理を妨げることがあるのは，EXECサービスの処理優先度がそれを要求するプログラムの優先度とは連動しないからである．各種EXECサービスの内，ファイルの割当/解放（オープン/クローズ）やプログラム・サイズの変更等の比較的冗長な処理を要する要求は，クライアント/サーバの形態で処理する．処理構造の単純化が図れるが，アクティビティの作成/消滅あるいはコンテクスト・スイッチを伴い，負荷が大きくなる（複雑で長い処理時間を要するサービスの場合は，負荷が大きい欠点は相対的に小さくなる）．

まず要求したアクティビティ（クライアント）を待ち状態にし，次にEXECワーカと呼ぶ低位EXEC優先度のアクティビティ（サーバ）を作成し要求を処理する．バッチ処理等の低い優先度のプログラムの要求をより高い優先度のEXECワーカが処理するので，結果的に要求を処理している間は優先度が上がることになる．したがって，EXECワーカより低い優先度の可変優先度トランザクションは，間接的に，バッチ処理の影響を受けることになる．逆に固定優先度トランザクションのプログラムが出せば結果的に優先度が下がることになる．しかし，トランザクション・プログラムが冗長な処理を必要とするような要求をすることは通常ない(また，その必要もない)．要求のほとんどは短い実行時間で処理できるものに限られる（たとえば，入出力処理や日時/時刻の取得)．そのような要求は，EXECモードと呼ぶ特殊なモード（優先度が0）で実行処理する．要求者のコンテクスト，しかし要求者（ユーザ）のレジスタとは別の専用レジスタ（EXECレジスタ)，で実行する（要求者レジスタのセーブが不要)．実行環境の切り替えに伴う負荷を最小限に押さえることができるので非常に効率の良い実行が可能である．実行の途中で割り込みが発生しても割り込みがあった旨を記すだけで直ちに割り込まれた処理に復帰し，処理を終えた時点で改めて割り込み処理を行う．全ての処理を終えると要求者に戻る．入出力要求の場合は，入出力の完了待ちとなるので，優先度の高い別のアクティビティを選択し実行することになる．

業務上の事情を抜きにすれば，固定優先度トランザクションが処理時間が短いものに向いているのに対して，可変優先度トランザクションは処理時間が比較的長くかつばらつきのあるものに向いている．SQLをトランザクション・メッセージとして受け付けて処理するプログラム（たとえば，ODBCサーバ）を可変優先度トランザクションとする，とかである．単に特定のレコードを検索する場合の秒単位の処理から，大規模データベースへの全件検索や結合・グループ化・ソート処理を伴う場合の分単位や時間単位の処理まで，と検索処理時間に大きなばらつきがあるからである．デスクトップ・ツールを使いメインフレーム上の膨大なオペレーショナル・データを検索し加工分析するような利用方法が今後ますます増えてくると思われるが，多様なスケジューリング方式が既存業務との無理のない同居を可能にする．

バッチやデマンドと同じ扱いを受けることになる時分割優先度は，バッチ処理等との共存を図る上で有用である．

ディスパッチャの多様性は，TIP スケジューラの多様性と相俟ってシステムに多面性を持たせ，多種多様なアプリケーションを効率よく実行することを可能にする．多面性は，<APG，入力メッセージの優先度，アクティビティの優先度>からなる三次元空間に由来する．使用者は，アプリケーションリをこの空間に適切にマッピングすることにより，システムの最適化を図ることができる．マッピング自体は，VTBUTL プログラムで設定する属性で決まるので，簡単に設定変更可能である．

4.3.3 時分割優先度クラス

バッチ，デマンドをはじめトランザクション処理（の一部）もこのクラスで実行するが，それらを区別することはしない．単に，動作特性の違いに基づく選別を多段帰還方式により行う．ただし，I/O バウンドのアクティビティの優先度の上昇をクラスの中間の相対優先度に止めるようにする．この結果，処理時間が短く相対優先度が中間より低くなる前に処理を終える（一般的にはデマンドやトランザクション処理）アクティビティはクラスの上半分の優先度域で実行し，処理時間が長い（バッチ処理）アクティビティは下半分の優先度域で実行するようになる．実質的には，二つのクラスと見なすこともできる．下半分の優先度域に着目すれば，I/O バウンドのアクティビティは高い優先度で，CPU バウンドのアクティビティは低い優先度で，実行する単純な多段帰還方式であり，スループットの向上を図ることができるが，一方では，CPU バウンドのアクティビティがなかなかサービスされない問題が生じる．このことは，一つのランが遅れるだけには留まらない．一般に，バッチ・ランには（遅れるのはバッチ・ランとは限らないが），先行ランや後行ランが存在し相互に処理ネットワークを形成しているので，一つのランの遅れはネットワーク全体の遅れにつながるからである．遅れが甚だしいと，翌日のオンライン業務の立ち上げが遅れることも考えられる．このような問題への対処として，低い優先度のアクティビティを定期的に順次優先度を上げて行くような方策を取り入れている．

5. おわりに

システム資源のスケジューリング機能を概観（構成法や実装技術を別に）すれば，メインフレームと UNIX との間に大きな差はない．それは，メインフレーム OS の考え方を取り入れている WindowsNT*[3] についても一般である．依って立つ基本理論，それらの多くは四半世紀以前に確立されている，が同じなので当然と言えば当然である．20 年程前の OS の解説書が格別陳腐化していないことにも現れている（そもそもその手の本は最近流行らないようだが）．しかし，それはあくまでも話の上でのことである．現実のシステムは別儀である．

メインフレームには基幹業務システムを担ってきた実績がある．社会的影響が大きい重大障害で新聞ねたになったこともある．ホットスタンバイ・システムが実際に機能し，事なきを得たこともある．所要のスループットに至らず負荷削減に苦闘したこともある．堅牢である，信頼性がある，あるいはトランザクション処理に強い，等のメインフレームに対する評価は，色々な意味で基幹業務システムで鍛えられてきた賜

である．しかし，メインフレームに冠せられるこれらの言葉には，具体性に欠ける恨みがある．本稿は，シリーズ2200を支える技術，とくにトランザクション処理を解説することで，シリーズ2200が如何に基幹業務システムに適合しているかに若干なりとも具体性を与えようとの試みであるが，成果の程はどうであろう．

シリーズ2200は，役目を終え引退したわけでも老境に入ったわけでもない．これからも進化を続けて行くことになる．それは，メインフレームとしてのさらなる洗練である．あるいは，従来の枠を超えてのオープン世界との連携融和である．CMOS技術によるコスト・パフォーマンスの飛躍的向上，XTPAによる高いスケーラビリティや高信頼性に加え，分散トランザクション処理機能を提供するTransIT*4 Open/OLTP*5，UNIXの開発実行環境を提供するOPE（Open Programming Environment）/やデスクトップ・ツールとのインタフェース（ODBC）を備えることで，（既存資産を継承しつつ）メインフレームから脱皮しオープン・エンタープライズ・サーバとして飛翔することである．それは，これまで以上の多様なアプリケーションの実行を意味し，標準的な優先度クラス構成に基づくスケジューリングだけでは不十分となることも考えられる．多くの場合は，優先度クラス構成をシステム生成で適宜変更することで対応可能であろう（柔軟なクラス構成が可能）．場合によっては，スケジューリング方式の手直し等の対処も必要となることも考えられる．また，XPC，キャッシュ・ディスクや半導体ディスクの普及に伴って入出力の多くが1ミリ秒前後ないしは数ミリ秒で終了するようになっている．単純にI/Oバウンドのアクティビティの優先度を上げる方式では，ますますCPUバウンドのアクティビティの処理が滞り，定期的な優先度の調整だけでは追いつかない可能性もある．入出力に要した時間を勘案して優先度を決める等の対処策を検討する必要があろう．

---

*1　UNIXは，X/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
*2　Machは，カーネギーメロン大学で開発されたOS名である．
*3　WindowsNTは，Microsoft社の商標である．
*4　TransITは，Unisys社の登録商標である．
*5　Open/OLTPは，Unisys社の登録商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

執筆者紹介　城　川　孝　二（Koji Shirokawa）
1970年防衛大学校航空工学科卒業．同年日本ユニシス（株）入社．EXECの開発・保守・性能評価を担当．現在，システムプロダクト部サーバソフトウェア開発室に所属．

Aシリーズにおける 大規模トランザクション処理を実現する要素技術

Technologies for High-Volume Transaction Processing on A Series Systems

森　良行

要　約　大規模トランザクション処理システムは，基幹業務系を司るシステムに適用されている．基幹業務系であるがゆえにトランザクション処理の応答時間は，一定時間内の枠に入っていなければならない．これを保証するためにシステムでは，業務処理上のさまざまな考慮がなされている．これらの考慮の前提になるのは，基盤システムが持つ処理の多重化を行う技術である．

Aシリーズでは，オペレーティング・システムであるMCP（Master Control Program）が提供している多重プログラミング環境やプロセッサ割り当ての優先度の制御，ディスクファイルの制御の技術を他の基盤ソフトウェアが有効に利用している．

たとえば，トランザクション処理システムのTPモニタとしてCOMS（COmmunication Management System）の動的リンクライブラリによる業務処理の並列化は，一連のトランザクション処理が各トランザクションプロセッサ上で行われるため，トランザクションプロセッサの数の並列処理を可能としている．また，データベース管理システムのDMS II（Data Management System II）において，データベースをアクセスするカーネルプログラムが動的リンクライブラリとして業務処理プログラムから呼び出されるため，稼働している業務処理プログラム数だけ並列処理ができるし，この並列性がデータベースの入出力操作にも適用できる仕組みを持っている．

本稿では，Aシリーズにおける多重プログラミング環境を提示し，その環境下でトランザクション処理システムを支える基盤ソフトウェアが行っている多重化処理の仕組みについて述べる．

Abstract　High-volume transactions processing systems are used in systems that govern mission-critical applications. Just because these applications are mission-critical, their transaction processing response time must be within a certain limit. In order to ensure this, these systems considers many factors regarding application processing. Prerequisite to this consideration is the process-multiplexing technology of the base system.

On A Series systems, the operating system or MCP (Master Control Program) provides multiprogramming environment, processor-allocation priority control, and disk-file control technology, which are effectively utilized by other pieces of base software.

For example, when using COMS (COmmunication Management System) as the TP monitor to use its dynamic link library for application processing, transactions of up to the number of the transaction processors can be processed concurrently because each transaction processor can perform a series of transaction processing. And DMS II (Data Management System II) also enables concurrent processing for each running application program because the kernel program actually accessing the database is invoked from each application program using dynamic link library, and this capability can be applied

to database I/O operations as well.

This paper presents the multiprogramming environments on A Series systems and describes the mechanism of multiprocessing performed by the basis software to support the transaction-processing system under this environment.

1. はじめに

近年，集中処理されていたトランザクション処理システムを分散することが注目され，分散処理技術がコンピュータ・システム上に適用されてきている．しかしながら，大容量のデータを持つ大規模トランザクションシステムでは，応答時間の保証や一日の処理可能なトランザクションの量，バッチ処理におけるデータ操作上の問題によって，分散トランザクション処理への移行が困難な状況にあり，メインフレームによる集中トランザクション処理が行われているのが現状である．

トランザクション処理システムは，通常基幹業務システムの中核をなしている場合が多い．このため，システムの稼働状態が悪化すると，業務が遅延したり顧客を待たせたりすることが発生し，結果的には，会社として不利益を被ることになる．このような状況を発生させないために，これらのシステムでは，コンピュータのリソースを効率良く使用し，システムに適した割り当てを行う技術が適用されている．

たとえば，数百数千のクライアントを持つシステムでは，各クライアントごとに業務処理プログラムを割り当てることは現実的には不可能であるし，クライアントからの処理要求は同時に発生するわけではない．このため，これらのシステムでは，TPモニタ（後述）を使用して業務処理プログラムをトランザクションプロセッサ（後述）として稼働させる形態，いわゆるOLTP（OnLine Transaction Processing）の形態のシステムを導入している．

本稿では，Aシリーズ上のOLTPの処理形態をもとに，コンピュータ・リソースの利用配分に焦点を当てて，オペレーティングシステムによる多重プログラミングの制御，TPモニタによるトランザクションプロセッサの制御，データベース管理システムの制御の仕組みを紹介する．

2. オンライン・トランザクション処理

2.1 トランザクション処理システムの機能

一般にトランザクション処理は，OLTPと呼ばれていて，処理単位としてACID特性を持っていることが前提となった処理を示す．ACIDは，以下の四つの特性の頭文字を取ったもので，Andreas Reuterが1983年に提案した言葉である．

原子性（Atomicity）：トランザクションが分割できない作業単位であることを意味する．一つのトランザクションに対して処理が正常に完了するか，または，処理が全て取り消されるかの二者択一の処理の単位であること．

一貫性（Consistency）：トランザクションの処理結果が一定していること．すなわち，トランザクション処理結果が論理的矛盾を起こさないこと．

独立性（Isolation）　：並列にトランザクション処理を実行している場合に相互のトランザクションが干渉しないでトランザクションとして独立していること．他のトランザクションによって処理結果が左右されないこと．

持続性（Durability）　：完了したトランザクションについては，障害によってそのトランザクションの処理が消滅することはない．すなわち，完了したトランザクションの情報がデータベース上に反映されることを保障すること．

ACID 特性を持つということは，トランザクションが回復・一貫性・並列性の基本単位となり，トランザクション処理システムが業務処理で期待される一貫性を保証することを意味する．

トランザクション処理システムでは機能別にみると以下のモジュールがあり，図1にその構成を示す．

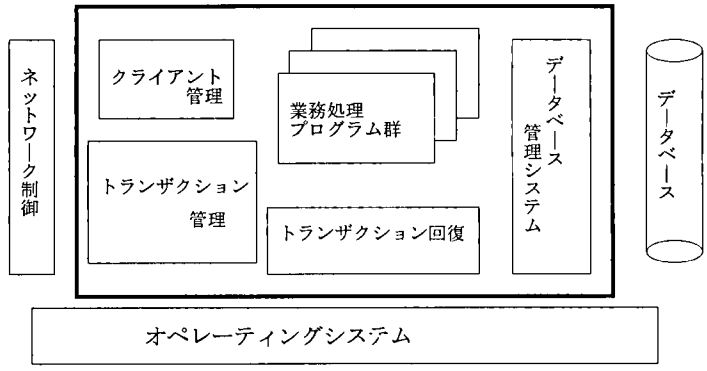


図 1　トランザクション処理システムの構成図

なお，以下の「クライアント」は，ワークステーション，PC，端末を含めた総称として使用している．

1)　業務処理プログラム群

業務処理プログラムは，業務やトランザクションの種類に応じて存在する．これらのプログラムは，トランザクション管理モジュールより自動的に起動され，各トランザクションは，トランザクション管理モジュールを介して業務処理プログラムに渡される．また，同一プログラムが複写されて多重に稼働することもある．これらの業務処理プログラムをトランザクションプロセッサ（TP）と呼ぶ．

2)　クライアント管理モジュール

このモジュールは，クライアントからのメッセージ形式の変換とクライアントとの送受信に関する操作を行う．この機能によって，トランザクションプロセッサは，扱うトランザクション形式の統一性が保証され，クライアントとの送受信にともなう物理的操作から解放される．

また，クライアント管理モジュールは，トランザクションプロセッサとクライアントとの送受信において入力トランザクションの保存，出力メッセージの到達

確認，未送信メッセージの保存と出力管理などの物理的な送受信を補う機能も持つ．したがって，トランザクションプロセッサは，業務処理と直接関係しないクライアント操作から解放され，送受信先を論理的に扱うことが可能となる．さらに，利用者がクライアントを介して扱える業務を保証するために，トランザクション単位での安全保護（セキュリティ）の管理機能も提供する．

3) トランザクション管理モジュール

このモジュールは，業務処理に応じてトランザクションプロセッサの起動・終了の実行を管理し，トランザクションプロセッサとクライアント間のトランザクションの経路管理を行うモジュールでトランザクション処理システムの中核をなす．このモジュールは，トランザクション管理（TM: Transaction Manager）プログラムとかTPモニタ（Transaction Process Monitor）とも呼ばれる．

4) データベース管理システム

データベースに対する操作や回復処理を行うモジュールで，OLTPシステムとしてトランザクションの特性を保証できる回復処理と，そのための業務処理インタフェースを持つ．

5) トランザクション回復モジュール

トランザクションデータのロギング（監査証跡）機能と，その監査証跡ファイルを使用した回復処理の機構を持つモジュールである．このモジュールは，データベース管理システムとトランザクションプロセッサ間のインタフェースを持ち，トランザクション特性を保証するためにデータベース管理システムの回復処理に対する補完機能を司る．

上記のようにOLTPのシステムは，トランザクションが保証する一貫性を業務処理プログラムに依存し，データベース管理システムがその機能を補完する．また，その前提となるトランザクションの整合性は，TPモニタが保証するといった機能のすみわけがなされている．これらのシステム構成上の機能分類は，UNIX*1やPC(パーソナル・コンピュータ）サーバであろうが，メインフレームのシステムであってもかわらないが，各システムの特性やOSの持つ機能によってTPモニタやトランザクションプロセッサが備える機能が異なっている．

2.2 Aシリーズにおけるトランザクション処理環境

Aシリーズにおけるトランザクション処理に関わるソフトウェア構成を図2に示す．OLTPシステム環境のソフトウェア構成との関係は表1の通りである．

トランザクション処理システムにおける処理の多重化を行う上で，トランザクション管理を行うTPモニタのCOMS，大規模データを取り扱うデータベース管理システムのDMS II，プロセスのスケジューリングを行うオペレーティングシステムのMCPが連携して効率良いシステムリソースの配分を行いトランザクション処理の効率を上げている．

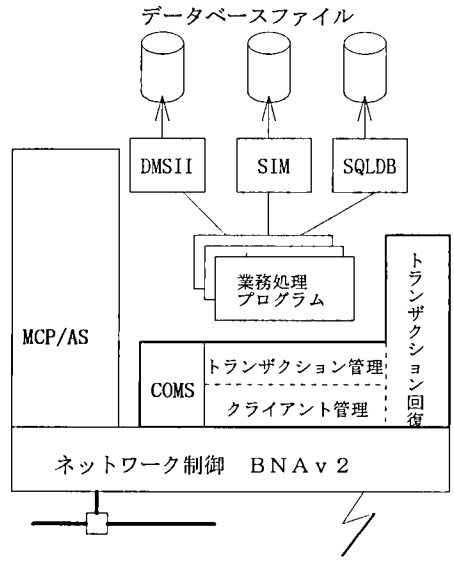


図 2　ソフトウェア構成図

表1　Aシリーズでの実装ソフトウェア

	Aシリーズでの実装ソフトウェア名
オペレーティングシステム	MCP(Master Control Program)/AS
ネットワーク制御	ASN(A Series Network) Aシリーズのネットワーク制御系のソフトウェアでLAN制御も司る.
トランザクション管理	COMS(Communication Management System) Aシリーズでは，TPモニタをメッセージ制御システムとも呼ぶ.
クライアント管理	実装のソフトウェア構成ではCOMSに組み込まれている.
トランザクション回復	実装のソフトウェア構成ではCOMSに組み込まれている.
業務処理プログラム群	COMSで実行管理されるトランザクション・プロセッサ (業務処理プログラム)
データベース管理システム	階層，ネットワーク型のDMS II，リレーショナル型のSQLDB，意味モデル型のSIMの三つのデータモデルがある.

3. オペレーティング・システム MCP の制御

3.1 プ ロ セ ス

Aシリーズで使用するプロセスとは，

- ジョブ制御言語であるワークフロー言語（WFL: Work Flow Language）で記述された仕事の集まりであるジョブ.
- ジョブの中で<タスク起動文>で実行されるタスク．このタスクは，通常，プログラムコード（業務プログラムなど）に対応する.
- プログラムの中より，特定の手続きを各種言語の持つタスキング機構によって起動されるタスク.

等を含めた対象を示す．Aシリーズでは，これらのプロセスに対してスタック（プロセス・スタックと呼ぶ）を割り当て，そのスタック番号によってプロセスを識別する.

一つのプロセスは，どの時点でも一つの指令しか実行できないし，その指令の参照する活動記録（プロセス・スタック）も一つである．

一般に一つのプログラムは，タスキング機構を用いることによって，ある時点で同時に複数の指令を実行することができる．この個数が実装しているプロセッサの数より多い場合に多重プログラミングとして制御する．Aシリーズでは，一つのプロセスから別のプロセスを生成する場合，もとのプロセスとデータの共有を行う場合には親子関係を持ったプロセスとして生成する．親子関係を持っているプロセスの集まりをプロセスファミリと呼ぶ．また，子プロセスが親プロセスの手続きを呼び出す場合は，子プロセス上でその手続きの処理を行う形態をとる．同様に，図3に示すようにプロセスが例えば入出力のためにMCPの手続きを呼び出す場合でも当該プロセス上で処理を行うし，プロセッサを解放するためのプロセス切り替え処理もMCPの手続き呼出しとして当該プロセス上で処理する．この様に，Aシリーズでは，他のプロセスで定義されたアルゴリズム（手続き）を利用する場合，プロセス切り替えを行わず，自分のプロセス内で定義した手続き呼び出しと同じ機構で処理できる．このため，プロセス切り替えによるシステム上の負荷が減少する．

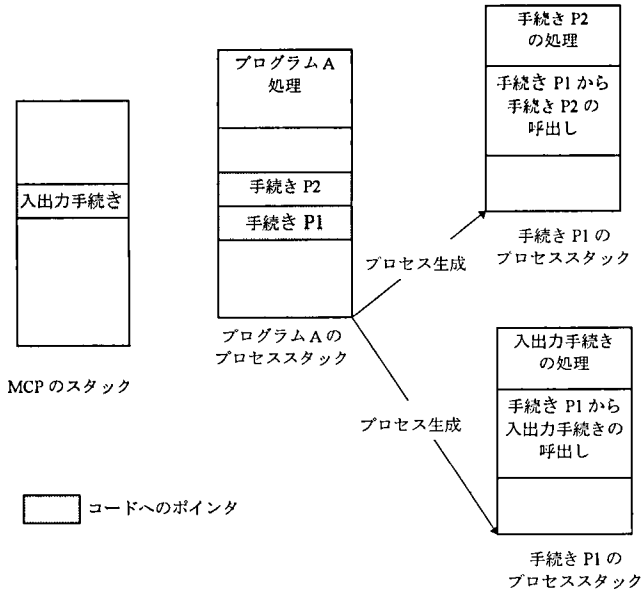


図3　他プロセスの手続き呼出し図

3.2　多重プログラミング環境での優先度制御

多重プログラミングの制御として，プロセッサをどのプロセスに割り当てるかといった課題と，一つのプロセスにどの程度プロセッサを占有させるかという課題がある．前者については，最初にプロセッサ要求を行ったプロセスが最初に割り当てられるのが一番単純なケースであるが，多重度の高いシステムであればプロセスが行う処理内容に優先度があるのでプロセス間でのプロセッサ優先度を考慮する必要がある．

MCPは，プロセスを生成する時，そのプロセスの属性を示す情報をプロセス情報ブ

ロックに設定する．利用者プログラムは，自分自身のプロセスや生成するプロセスのプロセス情報を参照したり更新するためにタスク属性と呼ばれる属性情報を介して，プロセス情報ブロックをアクセスできる．プロセス情報ブロックには，プロセッサ優先度が含まれていて，タスク属性を使用して優先度を設定することができる．

プロセッサ優先度には，システム稼働上でOSが設定する優先度（BIASと呼ぶ），利用者プログラムが稼働する上で用いる優先度（BASE PRIORITYと呼ぶ），プロセスの活動状態から設定される優先度（FINE PRIORITYと呼ぶ）がある．MCPは，プロセッサの待ち行列（Ready Qと呼ぶ）に繋げる時に，この三つの優先度を使用して待ち行列のどこにプロセスを繋げるかを判断する．

BIASは，OSレベルの処理を司るプロセス（MCPの子タスク）に対して高い優先度が設定される．利用者プログラムに対しては，MCPの持つ共有リソースを占有するために排他制御を行った場合や，MCPが提供している排他制御用のAPI（Application Interface）を使用して排他制御中にそれより優先度の高いプロセスが待ちになった場合には，動的に優先度を高く設定する．これは，BASE PRIORITYが低いプロセスが共有リソースを占有した場合に，他の優先度の高いプロセスにプロセッサが供給されなくなる事態を避けるために行っている．また，中断要求されたプロセスに対してもこのプロセスが持つリソースを解放させるために優先度を高くする．

BASE PRIORITYは，利用者が指定できる優先度であり，利用者プログラム間ではこの優先度によって比較される．また，この優先度は，コンソール上から変更可能である．FINE PRIORITYは，上記の二つの優先度が同じプロセス間での比較要素となり，待ち行列に繋げるときに，その時点のプロセッサや入出力時間の使用状況に基づいて設定する．

Aシリーズにおけるプロセスがプロセッサを解放する契機は，入出力要求や排他制御等のMCP手続き呼出しを行った時や，ハードウェア割り込みが発生した時である．この時点で再度プロセッサ待ち行列に繋げられ，優先度の判断が行われる．プロセス切り替えの契機となるMCP手続きは，プロセスが排他制御を行う上で他に優先度の高いプロセスが排他するためにプロセッサを待っていないかを確かめたり，入出力を行うのための待ちに入るためにプロセッサを解放させる．ハードウェア割り込みは，もともと多重プログラミングにおける制御の切り替えの技術である．この割り込みには一定時間経過したら割り込みをかける計時割り込み，プロセスが実行しているプロセッサとは別のプロセッサまたは入出力プロセッサが行っている外部処理との同期をとるための割り込みがある．プロセッサを使用しているプロセスが例外現象を起こした時の内部割り込みは，プロセス切り替えの契機とはならない．

Aシリーズでは，A 17シリーズの発表以降，上記のMCPによる優先度制御とプロセス切り替えの処理をすべて，タスク制御プロセッサ(TCP-Task Control Processor)という専用プロセッサにオフロードした．入出力完了に伴うプロセスの切り替えも入出力プロセッサとTCPの間で行い，MCPが関与することなく実現されていて，多重プログラミング制御のMCPの負荷削減と効率化を図っている．

3.3 ディスクファイル制御

大規模トランザクション処理では，入出力操作の回数を減らすことと，入出力操作

を行うための待ち行列での滞留時間，いわゆる入出力待ち時間を減少させることによって，業務プログラムで処理可能なトランザクション件数を増加させることができる．入出力操作の削減は，業務プログラムに依存する要素であるが，トランザクション処理におけるデータベース入出力に関しては，「5.2節 データベース操作の効率化」で述べることにし，ここでは，入出力操作のスループットを高める並列入出力処理と，キャッシュ・ディスクによる入出力時間の短縮について述べる．

Aシリーズにおけるディスクは，ディスクドライブに対して名前付けを行い，その名前によってMCPはドライブを選択する．この名前をファミリ名と呼び，同一名を複数ドライブに設定して，これらを一つのファミリとして取り扱うことができる．ファイルやディスク領域の管理は，ファミリ単位に行っているので，ファイルは，ファイル名とファミリ名で識別され，該当ファミリに属するディスクドライブの総容量まで使用できる．

ファイルの使用領域は，利用者が設定した領域長(AREASIZE)と領域数(AREAS)をもとに確保する．領域長は，ディスク上で連続した領域の長さであり，最大値は，一つのディスクドライブの容量となる．領域数は，指定した領域長を持つ領域の最大使用数であり最初から確保する数ではない．ただし，一つのファイルは1000領域を超して領域を確保することはできない．ファイルの領域は，該当領域上のレコード出力が行われたときに確保されるので，設定された領域数をすべて確保した時の容量のディスクドライブをあらかじめ用意する必要はない．ファミリへのディスクドライブの追加はコンソール上から動的に行うことができる．また，領域を割り当てるディスクドライブは，ラウンドロビン方式で該当ファミリの各ドライブに分散されるように選択される．この様にディスク領域の割り当てにおいて，利用者は，ファイルに対して領域長と領域数を設定し，ファミリの割り当てと空き領域長の監視を行えば良い．

大規模トランザクション処理システムでは，大容量データに対して多大な入出力操作が発生する．このため，入出力操作の並列性が求められる．ディスクドライブは，複数の入出力パスで接続され，そこに接続されるディスクドライブの集まりで一つのバンクを構成している．一つの入出力プロセッサには，複数のバンクが接続されている．一つのシステムでは，複数の入出力プロセッサを持つ．すなわち，システム構成として入出力対象のディスク装置には多岐にわたる入出力パスがある．一つのディスクに対する入出力は逐次処理しかできないが，複数のディスクにファイルが分散されていることで入出力処理は，少なくてもディスクへのパスの能力に応じて並列に処理できる．

しかしながら，ファイルのディスクドライブの分割や，ディスクドライブの並列入出力操作だけでは，入出力のスループットを高めることはできるが，ミリ秒単位の入出力時間を大幅に短縮することはできない．そのために，ディスク側にキャッシュを持たせたり，ドライブ単位でメモリをキャッシュディスクとして利用する機能が提供され，ナノ秒のオーダで入出力操作とすることが可能となった．ディスクのキャッシュ化は，キャッシュ上のデータの入力時間を短縮することで入出力サブシステムのスループットを桁違いに高める可能性がある．したがって，従来のように入出力完了割り込みに基づいてMCPが入出力完了処理をすることは，プロセッサ資源の業務プロ

グラムによる効率的な利用を阻害する．そのため，入出力完了処理も入出力プロセッサにオフロードし，それを契機に発生するプロセスの切り替え処理も，専用のタスク制御プロセッサにオフロードする機構上の変更をA 17以降行ったのである．(この仕組みについては，本号「大容量データ処理を実現する入出力機構」を参照)

4. TPモニタであるCOMSによる制御

4.1 COMS

Aシリーズの TP モニタは，COMS (COmmunication Management System) と呼ばれるソフトウェアである．COMS は，業務処理プログラムへのアプリケーション・インタフェース (API) を提供するとともに，トランザクション処理環境の設定情報に基づいて動的リンクライブラリ (DLL-Dynamic Link Library) を多用してトランザクションの経路制御を行う．

COMS は，実行されると複数の内部手続きを別プロセスとして実行し，もとの実行主体は，動的リンクライブラリになる．COMS の実行構造の概要を図4に示す．図4にあるように，COMS プログラムそのものは，内部手続きを子プロセスとして実行した後でライブラリ COMS として静的状態になり，子プロセスが機能別に処理を行う．Aシリーズのライブラリプログラムは，静的状態になるとプロセスとして稼働することがないので，プログラム障害等で COMS が終了することはなく，メモリ情報は保証される．このライブラリプログラムは，MCP やネットワークソフトウェアから共有の動的リンクライブラリとして呼び出されるし，子プロセスからも大域変数が参照されたり，動的リンクライブラリとして呼び出される．

図4において，アクティブ COMS は，TP モニタの監視および操作員インタフェースを行うためにある．DB コントロールは，データベース管理システムとトランザクシ

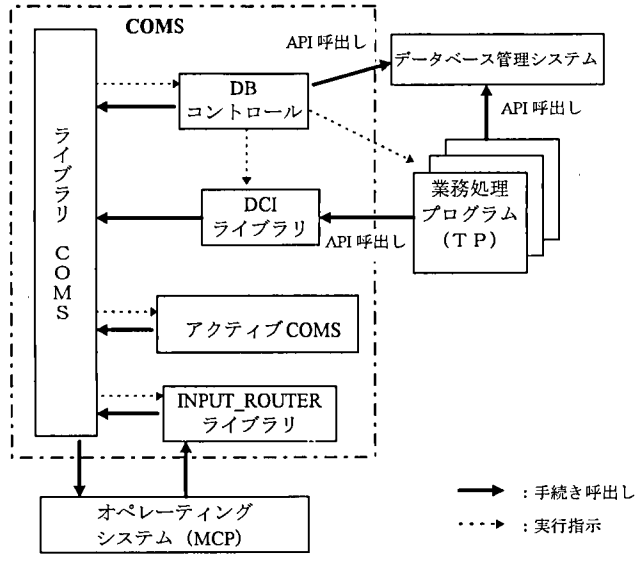


図 4　COMS の実行構造

ョン処理システムの整合性をとる役割を持ち，トランザクション処理で更新するデータベースごとに起動する．また，対象のデータベースを更新するTPは，このプロセスより起動され，このプロセスが起動したDCIライブラリを経由してトランザクションの送受信を行う．DCIライブラリも各TPの共有の動的リンクライブラリとなるので，各TPはDCIライブラリのAPIを呼び出すことで対象のトランザクションを送受信できる．

INPUT_ROUTERライブラリは，MCPの手続きがトランザクションを各TPの待ち行列に入れるために用意された動的リンクライブラリである．

COMSが制御する入力トランザクションは，ネットワーク制御系のプログラムがMCPの手続きを経由してINPUT_ROUTERライブラリを介してCOMSのシステムの待ち行列に投入され，各TPにより待ち行列から取り出される．各TPは，トランザクションをクライアントに送信する場合は，TPのプロセス上でDCIライブラリを介してMCP手続きを呼び出すことでネットワーク制御系のプログラムに渡すことになる．この様に，COMSはトランザクションの流れに対して動的リンクライブラリを提供することでプロセスの切り替えを最小に押さえている．

4.2 トランザクション処理の効率化

TPモニタには大量のトランザクションを効率的に処理することが求められる．従って負荷分散をどのように実現するかは重要な課題である．ここでは比較的一般的なパイプライン方式とCOMSが採用しているライブラリ並列方式を比較してみる．

1) パイプライン方式

COMS以前のメッセージ制御システムであるGEMCOS (Generalized Message COntrol System)で採用されていた方法で，GEMCOSを入力トランザクションの受付，経路選択，出力用などの機能別に分割し，それらを並列的に実行させる方式で，図5に示すとおり一定の単位時間で結果を得ることができる点で，プロセッサの高速化技術などにも用いられているパイプライン方式に似ている．

この方式をGEMCOSのトランザクション処理がProcess#1, 2, 3の三つのプロセスで分担して行う例で説明する．それぞれのプロセスの処理時間をt1，t2，t3とすると，各トランザクションプロセッサは，(t1+t2+t3)の時間ごとにトランザクションを受けるが，GEMCOSは，MAX(t1，t2，t3)の時間単位でト

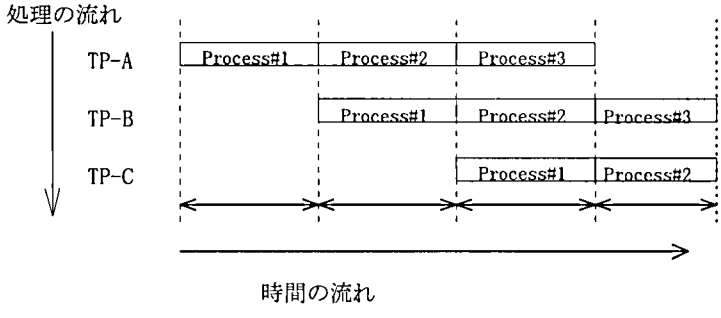


図5 パイプライン処理

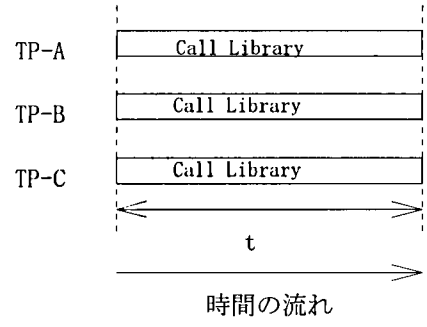


図 6　動的リンクライブラリ呼出の並列化

ランザクションプロセッサにトランザクションを渡せるので，GEMCOS のスループットを上げることが可能となる．

この場合，各パイプラインノード(Process＃1, 2, 3)当たりのサービス時間(tn)がある程度均一であることが望ましい．たとえば，トランザクション量増大に伴い，各パイプラインノードに待ちが発生した場合，一般的な待ち行列理論が示すとおり，そのノードが隘路になってスループットの低下が発生するからである．また，複数のプロセスが処理を担当するため，プロセス切り替えも必須となる．このようにパイプライン方式をソフトウェアの動作に適用した場合にはトランザクションボリュームに対する隘路が存在する．

2)　動的リンクライブラリの並列化

COMS は，パイプライン方式の隘路を図 6 に示すような動的リンクライブラリ呼び出しを並列化して用いることで解決している．

COMS のトランザクションプロセッサは，COMS の実行構造で示したように「RECEIVE」文によって DCI ライブラリという動的リンクライブラリを呼び出し，ネットワーク制御のプログラムから渡されたトランザクションを受け取る．その後，業務プログラムロジックを実行し，「SEND」文によって再び DCI ライブラリを呼び出し，ネットワーク制御のプログラムにデータを渡す．また，DCI ライブラリは，利用者記述の別の動的リンクライブラリを呼び出す機構を持っている．たとえば，業務処理的なデータの入力変換や出力時の画面フォーマッティングなどの RECEIVE の前処理や SEND の後処理を，トランザクションプロセッサとは別のモジュールとしてライブラリ化しておくと，DCI ライブラリがそれらを呼び出す機構である．したがって，待ち行列を介して処理する場合の待ち行列内の滞留時間や，プロセス切り替えのオーバヘッドがないように，COM は，トランザクションプロセッサの数だけ並列実行する機構になっているのである．

さらに，トランザクションの受取(RECEIVE)時点での待ちの解消という点では，COMS における各トランザクションプロセッサが最大 255 本までの同一コピーを持つことができることによって対処することができる．コピーの増減は，COMS がトランザクションプロセッサの入力トランザクション待ち行列の状態を判断して自動的に制御する．トランザクションプロセッサの数だけ COMS が

並列に実行できることによってパイプライン方式に見られた隘路というものは存在しなくなったと言える．

4.3 トランザクション処理環境の考慮

COMSは，複数の業務処理システム，複数のデータベースを制御することを前提にしている．このため，複数の業務処理システム間でリソースの競合による処理の遅延が発生しないようにトランザクション処理の環境が設定できる．

プロセッサの優先度は，COMS上に登録する各TPのプログラムごとに設定が可能なので，TPが処理するトランザクションの優先度を考慮して，TPのプログラムを分割しコピー数を適切に設定すれば，それらのトランザクションの処理の遅延は避けることができる．

障害回復を前提としてトランザクションの監査証跡の作成は，ディスクの書き出し完了を待つ必要がある．この処理に関しては，先に出力したものが先に処理されることになり，優先度を設定できない．このため，監査証跡の出力は，シーク時間を極力減少できるように専用のディスクを使うことができる．さらに，データベースごとに監査証跡を作成するので，データベースごとにディスクを割り当てることができる．このため，他の業務処理システムの監査証跡の書き出しのために処理を待たされることが避けられるし，監査証跡への出力のサービス時間は，ディスクの回転待ち時間とデータ転送時間に近い時間にすることができる．

5. データベース管理システムの制御

5.1 DMS IIの実行構造

Aシリーズのデータベース管理システムは，DMS II（ネットワーク型），SQLDB（リレーション型），SIMDB（意味型）の三種類があるが，実際の入出力操作やデータベースバッファの制御といった物理スキーマ部分については，すべてDMS IIの仕組みを用いている．このため，いずれのデータベースシステムを使用しても物理操作に対しては同じ考え方で対処できる．

DMS IIの処理構造は，COMSと同じように動的リンクライブラリによって業務処理プログラムへのAPI（Application Program Interface）手続きを提供している．

DMS IIは，データベース管理システムのカーネルプログラムであるアクセスルーチンによって，データベースの入出力の物理操作とバッファ管理を行っている．アクセスルーチンは，データベースに依存しない共通のプログラムであり，データベースごとに実行される動的リンクライブラリのプログラムである．DMS IIの実行構造を図7に示す．

図7にあるDMSUPPORTライブラリは，データベースごとに生成されるプログラムでデータベースの物理情報を提供するAPIや，COMSのトランザクション回復の再始動ポイントを管理するための制御を行うAPI等が実装されている．アクセスルーチンは，データベースの情報を得るためにDMSUPPORTライブラリのAPIを呼び出す．

業務処理プログラムは，アクセスルーチンが提供しているAPIを動的リンクライブラリとして呼び出すことによって，データベース操作が自分のプロセスとして行うこ

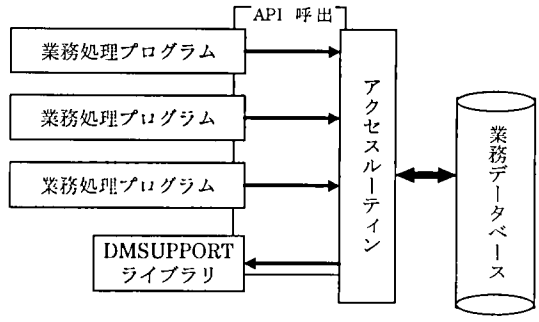


図 7　DMS IIの実行構造

とができる．アクセスルーチンは，操作員インタフェース以外には静的状態にあり，COMS と同様に動的な操作を行わない．このようにデータベース操作も業務プログラムの数だけ並列実行できる機構を提供している．

5.2　データベース操作の効率化

DMS IIが管理しているデータベースバッファは，データベースファイルの入出力操作用のバッファを代用していて，各構造体（データセット，セット）に対するバッファの割り当てが必要となった時点で動的に行う．DMS IIでは，データベースごとにデータベースバッファとして使用できるメモリ量の上限を設定するパラメタ(ALLOWEDCORE と呼ぶ)があり，この上限値を超えない限りどの構造体もデータベースバッファの割り当てが可能となる．

データベースバッファは，業務処理プログラムによって参照されたレコードやキーエントリ（索引テーブルのキー）を持つデータブロックやテーブル単位に管理されていて，参照を解除された時点でメモリ上より解放可能状態になる．実際のバッファの解放は．同じ構造体の読み込みバッファの割り当てがある場合に行い，参照解除時点では行わない．これは同じデータブロックやテーブルが再度参照される場合に再読み込みを避けるため行っている．しかしながら，同じ構造体の読み込みバッファの割り当てがしばらく行われないとバッファの解放が行われず，他の構造体用のバッファが割り当てできなくなることがある．このため，ALLOWEDCORE までデータベースバッファを使用している場合は，一定時間間隔で不要となったデータベースバッファをデータベースパラメタとして設定されている量だけ除去する仕組みを持つ．これにより，常に一定量のデータベースバッファが利用可能となり，かつ一定量のデータベースバッファをメモリ上に確保することができるので，参照頻度の高いデータは，常にデータベースバッファ上に存在することができる．また，これとは別に構造体として一度参照したデータブロックやテーブルをデータベースバッファ上に常駐させる設定も可能である．

DMS IIの業務処理プログラムのデータベースの更新処理は，図8に示すデータベース更新の流れのうちで

(1)　レコード移送－更新対象レコードを持つデータベースバッファを更新する．

(2)　更新情報の移送－更新情報をデータベース用監査証跡のバッファ上に設定す

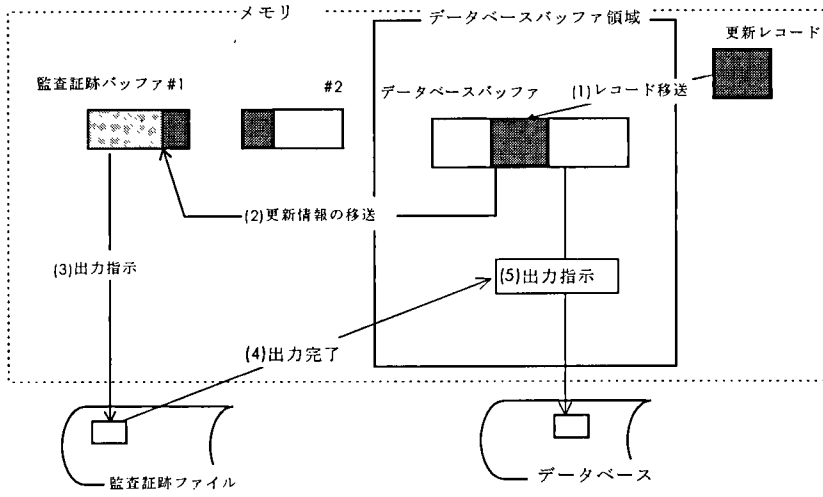


図 8　DMS IIのデータベース更新の流れ

る．

の二つの処理によって完了し，(3)の出力指示は，バッファが一杯のときにのみ行われ，(4)以降の処理は別の時点で行う．データベース用監査証跡は，トランザクションの監査証跡とは別に作成され，このバッファには，複数の更新情報が混在している．図8にある(3)の監査証跡の出力は，バッファが一杯になるか，回復処理用の制御点(SYNCPOINT と呼ぶ) になった場合に行われる．また，監査証跡のバッファは二つあるので，出力指示を行うだけで出力完了を待つのは次のバッファを出力するときに行う．これにより，監査証跡の出力完了を待つ時間が短縮できる．ただし，SYNCPOINT での出力時には回復処理に備えて完了を待つことになる．

更新したデータベースバッファは，その更新情報を持つ監査証跡の出力が完了するまでディスクへの出力ができない様に制御されている．これは，障害時にデータベースが更新されていて監査証跡上にその更新情報がないために回復できなくなることを避けるためである．データベースバッファの出力指示は，同じ構造体の入力指示（読み込み）を行ったときに入力完了を待つ間に行い，出力完了は次の入力時に確認するか，回復処理のための書き込み保証の時点に出力する．

また，DMS IIの機構として，構造体を順次参照していることを動的に判断し，参照対象のデータブロックやテーブルをディスクより読む場合に続けて次のデータブロックやテーブルの読み込み指示を行う先読み機構や，データブロックを連続する複数ブロックにまとめて一回の入力指示で読み込む再ブロック化機構がある．この様にDMS IIは，メモリ上のデータベースバッファを有効に使用して入力操作を減少させ，入出力操作を，業務処理プログラムと並列に処理できるようにして，かつ出力操作が集中しないように平準化できるように考慮されている．

6. おわりに

大規模トランザクション処理を行う上で必要となる多重処理と効率的な入出力操作に対して，Aシリーズが持つ機構の概略を提示した．Aシリーズでは，汎用機の障害に対する頑強性や信頼性，処理の効率性や多重度が期待される基幹業務システムが集中処理として稼働している．しかしながら，コンピュータ業界の方向としては，集中から分散処理へ向かっていることも否定できない．さらに，オフィス環境の電子化に伴い基幹業務の流れも変わりつつある中，新たな技術が要求されてくることになる．その一つに，オブジェクト指向を基本とした分散オブジェクトの技術がAシリーズ上でどのように実装され，トランザクション処理にどのように適用するかがあると考える．Aシリーズがオープン・エンタープライズ・サーバとして位置づけられて，メインフレームの良さを残しながらオープン化技術を実装していくことで分散処理への適用を期待できると考える．

---

*1 UNIXは，X/Openカンパニーがライセンスしている米国ならびに他の国における登録商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

参考文献 ［1］ Essential Client Server Survival Guide, Robert Orfail Dan Harkey Jeri Edwards 1994 Van Nostrand Reinhold, A Division of International Thomson Publishing Inc.
［2］ Aシリーズ・オペレーティング・システム―構造と制御―日本ユニシス(株)1992.

執筆者紹介 森　良行（Yoshiyuki Mori）
1951年生．1975年上智大学理工学部数学科卒業．同年日本ユニシス(株)入社．金融機関の客先サービスに従事した後，Aシリーズの利用技術サービスに従事する．現在，システムプロダクト部OES開発推進室に所属．

大容量データ処理を実現する入出力機構

I/O System Implementing High-Volume Data Processing

白崎　宏, 新津昭義
坂本徳明, 金井秀量

要　約　メインフレームを中核としたシステムはクライアント・サーバ・システムと比べて入出力分野が特に強いと言われている．本稿ではこの入出力性能および信頼性において圧倒的優位を誇るメインフレームの入出力機構について記述する．はじめにトランザクション処理における入出力の応答性能および多重度の観点から，入出力性能のシステム全体の性能における重要性について考察する．次にメインフレームの入出力を代表する先進的な入出力機構；シリーズ2200のXPCおよびAシリーズの機能別入出力プロセッサのアーキテクチャ，およびメインフレームのOSが採用している入出力処理技術について詳しく述べる．また，様々な入出力装置を接続するためにメインフレームがサポートする各種チャネルを紹介する．

Abstract　Mainframe-based systems are said to be more powerful in comparison with client-server systems, especially in terms of I/O capability. This paper describes the overwhelming superiority of mainframe I/O systems in terms of both their I/O capability and reliability. It first mentions to transaction processing from the viewpoint of I/O response time and multi-processing capability in order to see the importance of I/O performance as part of overall system performance. It then explains the typical advanced I/O architectures used in mainframes; the architectures of 2200 Series XPC and A Series function-based I/O processors, as well as the I/O processing software techniques used in mainframe operating systems. It also introduces the various I/O channel types mainframes support to connect different I/O devices.

1. はじめに

システム性能とI/O性能

メインフレームを用いた集中処理型のシステムは，クライアント・サーバ・システムと比較して，バッチ処理や大量のトランザクション処理の上で圧倒的な優位性を保持している．

バッチ処理は基本的に順次処理であり，アプリケーションの変更なしには分散化が難しい処理である．多くの場合，事後処理など一定時間内に終了することが要求される．また，トランザクション処理は高いシステムスループットを必要とする処理であり，ユーザから見て応答性が要求される処理である．

これらの処理は，いずれも入出力処理の占める割合がかなり高く，高速なCPUだけでなく，大容量で高性能な入出力機構を必要とする．実際，トランザクションの処理時間の7割近くは入出力処理に費やされており，処理の高速性，応答性は，システムの入出力処理能力に負うところが大きいといえる（図1）．

トランザクション処理におけるシステムのスループットは，トランザクション処理

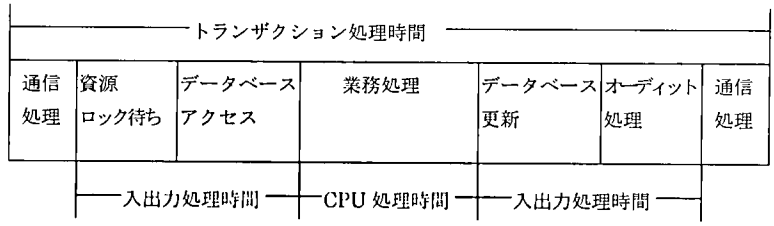


図 1　トランザクションモデルの処理時間分布

の応答性と多重度によって決まる．個々のトランザクションの応答時間と，並行して処理できるトランザクションの多重度は，CPU 性能やシステムリソースの量，データベースアプリケーションの形態などの幅広い要素に依存するが，特に，処理時間の大部分を占める入出力性能への依存度が高い．このため，個々の入出力について応答時間を短縮し，さらに多重度を高めることがシステムのスループットの向上につながる．

トランザクション処理の応答性

CPU，バスおよびメモリの高速化とともに入出力においても光チャネルの採用，大容量キャッシュメモリの搭載などの高速化が進んでいる．しかし，これらの技術を駆使してもデータの入出力にはミリ単位のアクセス時間および転送時間を要し，ナノ秒単位でデータアクセス，命令の実行を行う CPU，メモリの速度には遠く及ばない．

このため，入出力の比重が大きい処理においては，CPU 性能をいくら高めても入出力の完了を待つための遊休時間が増大し，CPU 性能の向上がシステム全体の性能向上につながらない．

すなわち，システム全体の性能向上のためには，CPU 性能の向上に見合った入出力装置の性能の改善あるいは入出力処理効率の改善が必要である．たとえば，あるトランザクションモデルでは，キャッシュメモリを持たないディスク装置にデータを乗せた場合は秒 100 件ほどを処理するだけで限界であったが，大容量キャッシュメモリを搭載したディスクを使用することにより秒 400 件以上もの処理ができるようになった．

このようなハードウェア高速化技術の採用とともに，ファイルの分散配置，あるいはファイル代わりにメモリをデータ格納庫として利用するなど，ソフトウェア機能，およびシステムの利用技術の面で入出力処理の効率化を行い，個々の入出力の応答時間を短縮することによってトランザクション処理の応答性を向上させることができる．

トランザクション処理の多重度

トランザクション処理の応答時間を短縮しても，システム内に滞留するトランザクション間でのシステム資源の競合や，大量の入出力発生にともなうオペレーティングシステムのオーバヘッドの増加などによって，システムのスループットがあがらない場合がある．

先に提示した図1のトランザクションモデルでは，トランザクションデータの処理は次のように行われる．

まず，トランザクションがデータ通信，メッセージ制御を通過してアプリケーションプログラムに渡されデータベースへのアクセスが発生する．このとき，業務処理を行ってデータベースを更新するまでの間に他のタスクから更新されないようにロックを行う．さらに，業務処理を行いデータベースレコードの更新を行って，オーディット処理が行われた後で，来た道の逆をたどって結果が返される．

複数のトランザクションが同時に処理されている場合には，データ通信，資源のロック，データベースが行う物理的な入出力など各所で競合が起こる．

このように，トランザクションデータがシステム内を流れていく先には，ハードウェア，ソフトウェアを問わず競合する部分があり，それの処理能力以上のデータが流れ込んだ場合，未処理のデータが蓄積し，レスポンスの低下を招く．

したがって，トランザクション処理能力を向上させるには，トランザクションの応答性とともに，トランザクションが多重化した場合の競合やオーバヘッドを減少させることと，トランザクション量が増大した場合に，隘路となりうる部分の強化が必要である．

すなわち，プロセッサ能力の向上，入出力プロセッサ能力の向上，チャネル処理能力の向上，さらには，入出力処理に関わるオペレーティングシステムの効率化や，入出力プロセッサへのオフロードなどによって，トランザクション処理の多重度を向上させることができる．

本稿の内容

トランザクション処理を中心に入出力の応答時間の短縮と入出力の多重度が増大した場合にも対応できる入出力性能の向上の重要性について説明してきたが，本稿では，入出力装置の応答時間をあげる技術としてシリーズ2200の拡張データ処理装置XPCについて記述し，入出力の多重度増大に伴う負荷をオフロードする技術としてAシリーズにおける機能別入出力プロセッサについて記述する．

また，入出力処理技術を支えるソフトウェア技術として，メインフレームのオペレーティングシステムが入出力性能向上のために行っている入出力処理について記述する．

さらに，従来の資産を継承しつつ新しい技術を取り入れていくために，メインフレームが支援している広範囲な種類のチャネルについても記述する．

2. メインフレームの入出力処理機構

2.1 XPC

2.1.1 XPCの概要

シリーズ2200において大容量データ処理を実現する入出力機構の中核となる拡張データ処理装置XPC（eXtended Processing Complex）について説明する．XPCは，非常に大規模なデータ処理/トランザクション処理を大幅に高速化する拡張処理アーキテクチャ2200/XPA（eXtended Processing Architecture）を実現するために登場

した，当社の新テクノロジをベースにした装置である．最大8台の独立したITASCA 3800，2200/900あるいは2200/500シリーズのホストから1台のXPCを共有して使用することをとおして，最大64台のCPU（Central Processor Unit）によるビジネス並列処理システムが実現され，新しいデータ処理環境における大規模サーバが構築される．

XPCは，2200/XPAを構成する四つのサブアーキテクチャのうち，入出力処理の性能の大幅な向上を目指したXIOA（eXtended Input/Output processing Architecture）を実現する装置であり，また大規模・高速トランザクション処理および無停止連続処理を目指したXTPA（eXtended Transaction Processing Architecture）を拡張・強化する装置でもある．

1) XIOA

XIOAは，トランザクション処理，データベース処理，バッチ処理，デマンド処理などの効率を高めるために，システムの高性能化に対応して，大容量記憶域上の共用ファイルおよび非共用ファイルに対する入出力処理の性能を大幅に向上させることをねらいとしている．大容量記憶域として，磁気ディスク装置，キャッシュディスク装置，キャッシュディスク・アレイ装置，半導体ディスク装置などが提供されているが，容量・価格・アクセス時間などの要因をすべて満足する装置はなくXIOAを実現するには至っていない（図2）．これがXPCのVSM（Virtual Storage Manager）機能によって初めて実現される（2.1.3項の1参照）．

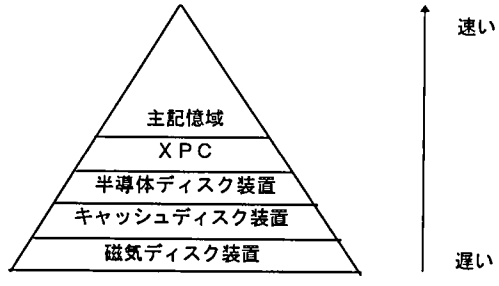


図 2 アクセス時間の比較

2) XTPA

XTPAは，最大8台の独立したホストシステムがデータベースをレコード・レベルで共有することによって結合され，各ホストシステムは本番稼働しながら，相互にバックアップし，大規模・高速トランザクション処理と無停止連続処理とを同時に実現させることをねらいとしている．従来はRLP（Record Lock Processor）を使用してXTPAは実現されていたが，XPCのDSM（Distributed System Manager）機能を使用することによりさらに拡張・強化される（2.1.3項の2参照）．図3にXPCを導入したシステム構成例を示す．

2.1.2 XPCのテクノロジ

1) XPCハードウェア

XPCは新テクノロジをベースに，XPC内の各コンポーネントを3次元のメッ

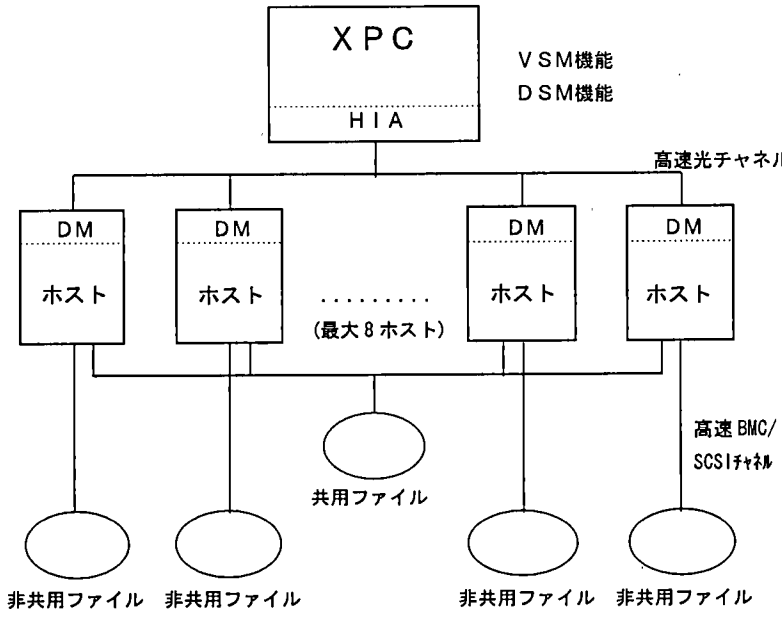


図 3　XPC を導入したシステム構成例

シュ構造で相互に接続して，高性能マイクロ・プロセッサによる高速並列処理を実現している．また XPC 自体の高速処理能力を生かすために，ホスト側の DM (Data Mover) と XPC 側の (Host Interface Adapter) を高速光チャネルで接続することにより 50 メガバイト/秒の高速並列データ転送を実現している．これは従来の高速 BMC (Block Multiplexor Channel) チャネル，SCSI (Small Computer System Interface) チャネルなどによるデータ転送をはるかにしのぐ処理速度である．さらに大容量データを処理するにあたり，データを完全に保証するため，XPC 内の各コンポーネントを二重化し，特に XPC メモリ上のデータに対しては，ミラー化およびメモリ・エラー自動修正機能など，より完全性を保持するための機構を備えている．

2)　オペレーティングシステムの対応

2200 オペレーティングシステム (OS) においても XPC の高速並列処理・高速並列データ転送を実現するための対応がなされている．

■入出力命令の実行処理

OS は XPC に対する入出力命令の実行を，命令の開始番地を主記憶域上のテーブルに設定するのみで完了する．DM は非同期にそのテーブルを参照して入出力命令を取り上げ XPC に渡し，XPC はそれを実行する．したがって OS は，XPC が入出力命令を受け取るまで待つ必要はなく，入出力命令をテーブルに設定すれば直ちに次の処理を実行できる．また OS は 1 台の DM あたり同時に最高 15 個の入出力命令をテーブルに設定するので，これにより並列処理はさらに加速される．

■入出力命令の完了処理

入出力命令の完了においてDMは，XPCから渡された入出力ステータスを保持するパケットの番地を，主記憶域上のテーブルに設定するのみである．割り込みの発生はなく，OSはシステム・アイドル時にテーブルを参照して入出力ステータスを検査する．従来の入出力装置においては，入出力の完了を割り込みによりOSに通知している．そしてOSは割り込み処理を最優先で行うため，そのとき実行中のアクティビティを中断し環境を保存してから，割り込み処理を行い，割り込み処理完了後に実行していたアクティビティの環境を再設定して制御をもどしている．しかしXPCでは割り込み処理はなくなるので，実行中のアクティビティの環境保存および再設定といったOSのオーバヘッドは一切なくなる．

■その他の処理

大容量記憶ファイルに対する入出力要求において，使用者プログラムはファイル相対番地を指定することにより，OSがファイル相対番地をもとにそのファイルが存在する装置およびブロック番号を割り出し，入出力命令を実行している．しかしXPCに対する入出力命令は，ファイル相対番地のまま実行するので，装置およびブロック番号の割り出しという処理が削減される．

上記3処理の対応により，XPCに対する入出力要求処理負荷数は他の大容量記憶域に対するものと比べ，大幅な削減となっている．

2.1.3 XPCの機能

XPCは2200/XPAのサブアーキテクチャXIOAを実現し，XTPAを拡張・強化すると説明したが，それに対応するXPCの主要な機能として，XIOAに対してはVSM機能，XTPAに対してはDSM機能が提供される．

1) VSM機能

大容量データ処理を実現する入出力機構として，データ入出力処理の高速化をはかるVSM機能は，①グローバル・キャッシュ機能，②プリフェッチ機能，③レジデント・ファイル機能，の三つに大別される．

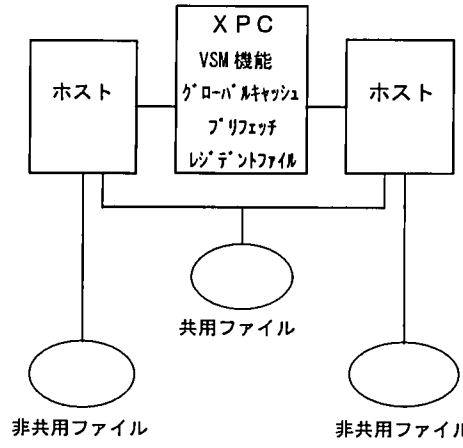


図 4 VSM機能使用構成例

図4にVSM機能使用構成例を示す．

① グローバル・キャッシュ機能は，アクセス頻度の高い大容量記憶ファイルのデータを，不揮発性でミラー化されたXPCの大容量メモリ上に保持する機能である．その結果，ホストから要求されるデータの読み込み/書き出しは，大半がXPCメモリ上で処理されることになり，大容量記憶域への実際の入出力回数は大幅に減少し，入出力処理の高速化が実現される．

XPCメモリ上のデータは1792語（1語＝36ビット）単位に“セグメント”として管理される．セグメントはデータに関する必要な情報，ファイル識別名/ファイル相対番地/ファイルが存在する大容量記憶域の識別名/ファイルが存在する大容量記憶域の装置相対番地，を保持している．なお入出力はセグメント単位である必要はなく1語から可能である．

グローバル・キャッシュ機能を使用すると，大容量記憶ファイルに対する入出力は，まずXPCに対して実行される．XPCに対する入出力において，必要なデータがXPCメモリ上にある場合を“ヒット”といい，この場合は大容量記憶域への入出力は行われず，主記憶域とXPCメモリ間のデータ転送で入出力が完了する．ヒットの場合のアクセス時間は，転送するデータ量に依存するが平均は1ミリ秒である．これは既存の大容量記憶域に対するアクセス時間と比べ，最も速い半導体ディスク装置よりも数倍速く，通常の磁気ディスク装置よりは数十倍速い．書き出しにおいては，例外を除き大容量記憶域への入出力は行われず，主記憶域からXPCメモリへのデータ転送で入出力が完了する．したがってオーディット・データのように書き出しを連続する入出力ファイルに対しては，入出力処理の高速化が顕著となる．図5に，入出力においてXPC上でデータがヒットした場合の概念図を示す．

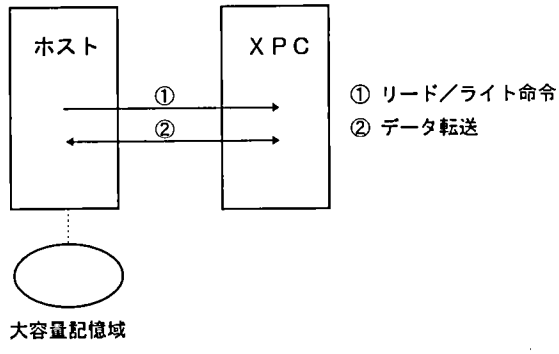


図5　ヒット時のデータの流れ

一方，XPCに対する入出力において，必要なデータがXPCメモリ上にない場合を“ミス”といい，この場合は次に説明するプリフェッチ機能により，大容量記憶域からXPCメモリ上にデータを転送し，その後XPCメモリから主記憶域へデータ転送を行い入出力は完了する．ここで大容量記憶域からXPCメモリ上へデータ転送する処理を“ステージ”という．

② プリフェッチ機能は，大容量記憶ファイルからの読み込みにおいて，XPC メモリ上にデータが存在しない場合（ミス）に，その読み込みで必要なデータに加え，今後必要となるであろうと想定される後続のデータについても，前もって大容量記憶域から XPC メモリ上に転送する機能である．その結果，ホストから要求されるデータの読み込みは，大半が XPC メモリ上から行われることになり，大容量記憶域への実際の入出力回数が削減し，入出力処理の高速化が実現される．このことからプリフェッチ機能は，順アクセスでファイルの読み込みを行う場合に非常に効果がある．図 6 に，入力において XPC 上にデータがなく大容量記憶域からプリフェッチ機能によりステージした場合の概念図を示す．

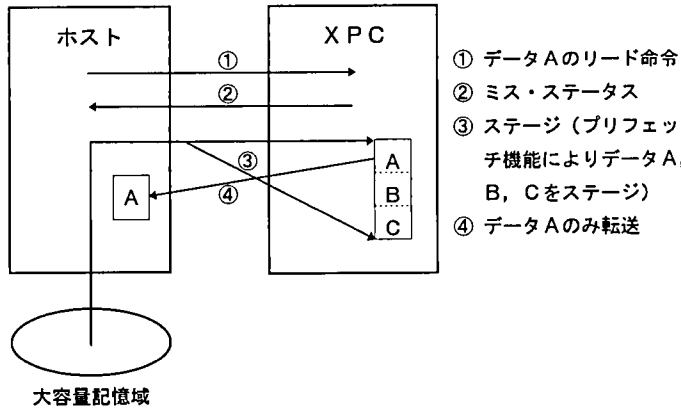


図 6 リード・ミスとプリフェッチ機能におけるデータの流れ

③ レジデント・ファイル機能は，グローバル・キャッシュ機能を発展させたもので，アクセス頻度の非常に高いテーブルやファイルを XPC メモリ上に常駐させる機能である．その結果，ホストから要求されるデータの読み込み/書き出しは，すべて XPC メモリ上で処理されることになり，大容量記憶域への実際の入出力が一切なくなり，入出力処理のさらなる高速化が実現される．

上記 3 機能は，基本的に使用者プログラムを変更することなしに，使用可能である．現行のシステムに XPC を導入し，システム生成時に VSM 機能使用の指定をするのみで，そのシステムのすべての大容量記憶ファイルはグローバル・キャッシングされる．これらの機能を組み合わせて使用することにより，XPC は大容量データ処理を実現する入出力機構の中核となるものである．

2) DSM 機能

大規模・高速トランザクション処理および無停止連続処理を目指した DSM 機能は，①レコード・ロック制御機能，②ホスト間メッセージ通信機能，の二つに大別される．

図 7 に DSM 機能使用構成例を示す．

① レコード・ロック制御機能は，共用データベースを複数のホストから参照できるようにするため，XPC メモリを使用してホスト間でレコード単位にロック

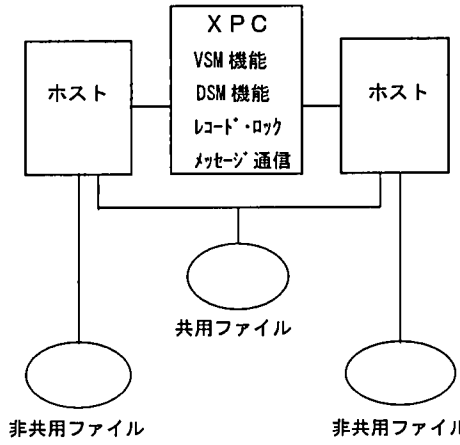


図 7　DSM 機能使用構成例

をかけることによりデータの同期をとる機能である．従来の RLP と比べ，ロック容量は XPC メモリを使用することにより最大 800 万件（16 倍）に拡張され，ロック能力は XPC の最大 32 個の演算処理エンジンが並列に処理することにより 6 万件/秒（6 倍）に向上する．高速・大容量のレコード・ロックにより大規模なトランザクション処理が可能となる．

② ホスト間メッセージ通信機能は，マルチホスト・ファイルシェア環境を制御するため，ホスト間のメッセージ通信を XPC を介して行うことにより，通信速度および安全性を向上させるものである．この機能をとおして各ホスト間で相互監視（ハートビート）を行い，障害の発生したホストをいち早く検知して回復処理を行い，システムの無停止連続運転が可能となる．

2.1.4 XPC の優位性

最後に XPC の優位性をまとめると次のようになる．

1) 導入・使用の容易性

最大 8 台の独立したホストから 1 台の XPC を共有でき，また VSM 機能および DSM 機能の両方が使用可能である．VSM 機能においては各ホストにいかなる種類の大容量記憶域が接続されていようと，ファイルを乗せ替える必要はない．システム生成において VSM 機能を使用する指定をすれば，全ファイルともグローバル・キャッシングされる．また使用者プログラムを変更することなく VSM 機能は使用可能となる．

2) ハードウェアの高信頼性・耐障害性

XPC を構成するハードウェア・コンポーネントは新テクノロジをベースに完全に二重化または多重化されており，障害に強い構造となっている．したがってデータも完全に保全する機構となっている．

3) 入出力の高性能性

VSM 機能を使用することにより平均 1 ミリ秒というアクセス時間で入出力は完了するので，システム全体の入出力処理は飛躍的に改善される．

4) システムの無停止性

DSM機能においては，複数ホストで稼働中に万一いずれかのホストで障害が発生しても，障害の発生をいち早く検知してそのホストを自動的に切り離し，他のホストが障害ホストの処理を引き継ぐので，システムは何の影響を受けることなく継続し無停止連続運転が実現する．

2.2 機能別入出力プロセッサ

2.2.1 Aシリーズの大量入出力処理技術

Aシリーズが採用している大量の入出力を処理する技術は，入出力時間を短縮し高速化する技術と，その結果，大量に発生する入出力開始・終了の入出力制御処理の処理性能を大幅に拡大する二つの技術から成る．前者は，ミリ秒のディスクアクセス時間をメモリアクセス時間に短縮させ，システム全体としての処理性能を格段に高める「キャッシュディスク」機能によって実現し，後者は，機能別入出力処理専用プロセッサによって実現している．

キャッシュディスク機能は，入出力バンド幅を最大化し，大規模トランザクション処理での入出力時間を短縮させるものであり，以下の3分野において実現している．

1) メモリディスクの実現

この機能は，ディスク装置そのものをメモリ内に実現するものである．メモリ内のディスクであるため，メモリディスクに格納されたファイルへのアクセスは，通常のデイスク装置に配置されたファイルと比べアクセス時間が大幅に短縮される．メモリディスクに対する操作は通常のディスク装置への操作と同じであり，この機能の利用には，業務プログラムやジョブ制御言語への変更を必要としない．

2) グローバルディスクキャッシュの実現

Aシリーズでは，ディスク制御装置やディスク装置でキャッシュ化するローカルディスクキャッシュに加え，メモリを用いた「グローバルディスクキャッシュ」も実現している．この機能は，入出力指令のチャネル上のトラフィックの削減と，ディスク装置へのアクセス特性に基づいて，任意のディスク装置を任意の時間帯でキャッシュ化できる機能を提供する．

3) 目的別専用のディスクキャッシュの実現

ギガ，あるいは，テラのオーダのディスク容量を，メガや数ギガのオーダのメモリに効率的にキャッシュ化するには，そのシステムで稼働している業務プログラムからのディスクアクセス特性を反映したキャッシュ制御が望ましい．そのために，Aシリーズでは，目的別の専用キャッシュとして，データベース専用キャッシュディスクとオペレーティングシステムの使用する構造体のための専用キャッシュディスクを実現している．

データベース専用キャッシュディスクは，データベース固有の動き，すなわち，セットやデータセットごとのアクセス方式や先読みの可否，用意すべきバッファ数などに基づいて，セット，データセット単位で最適な必要メモリ量を確保するのに用いられる．一つのデータベースで1.5ギガバイトまで利用できる．また，構造体キャッシュは，業務プログラムから頻繁にアクセス要求のあるディスクファイルアクセス用の各種の構造体，コードファイル管理用の構造体，セキュリテ

ィ用の構造体などのための専用キャッシュディスクである．

以上の3分野でのキャッシュディスク技術の実装に加え，ディスク上のデータの保全性を高めるミラーディスク機能も実装している．これは，全く同じ情報を保持する複数のディスク装置を生成・維持するもので，オペレーティングシステムによるRAID 1の実現でもある．すなわち，ディスクサブシステムにRAID 1を搭載していないディスク装置でもミラーディスクとして使用することができ，データの保全性が保証され，また，読み取りに関しては，複数のディスク装置に負荷分散されるため読み取り時間の短縮にも役立つ．Aシリーズでは，キャッシュディスク機能とミラーディスク機能を併せて「高速入出力支援ファシリティ」と呼ぶ（図8）．

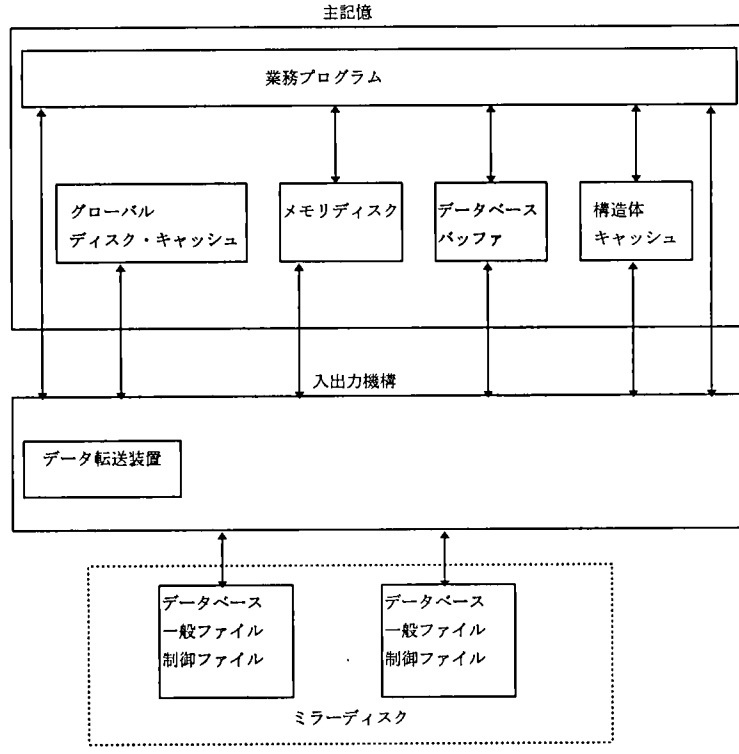


図8　Aシリーズ高速入出力支援ファシリティ

これらの入出力機構の高速化技術の中で，機能別入出力プロセッサは，トランザクション処理の多重度を向上させるものである．従来，Aシリーズのオペレーテイングシステムである MCP（Master Control Program）が実行していた機能の一部を，独立したプロセッサにオフロードすることによりシステムスループットを向上させている．Aシリーズが搭載している機能別入出力プロセッサについて，その背景と実装技術を次に述べる．

2.2.2　入出力処理におけるオペレーティングシステムのオーバヘッド

一般的に，オペレーティングシステムが行う入出力の役目は，業務プログラムから入出力要求を受け取り，それを適切な入出力デバイス制御装置に送ることと，入出力の終了処理を行い，入出力の終了を待っている業務プログラムを実行可能にすること

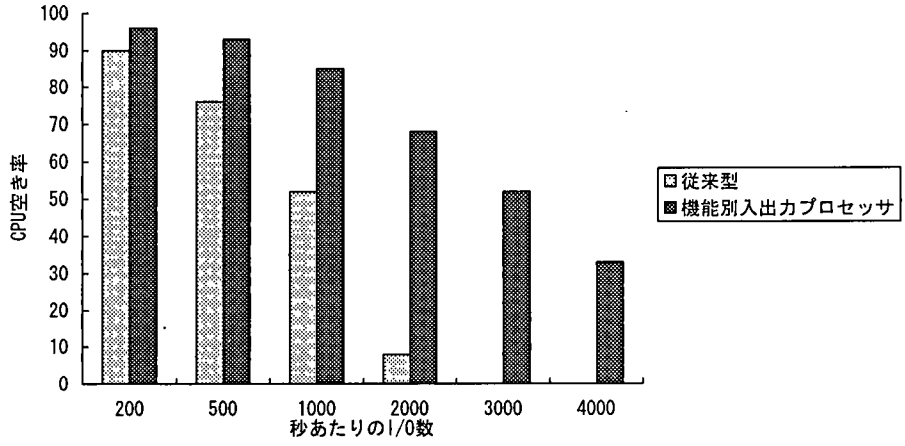


図 9　I/O 件数に対する CPU 利用率の変化

である．これらの処理はさほど大きな負荷になるものではないが，入出力要求の数，つまりトランザクション数が多くなるにつれて，オペレーティングシステムが実行するこれらの処理も増加する．さらに入出力多重処理のための共有データ等の排他制御による処理遅延も発生するため，これらの負荷はシステムスループットに大きな影響を与えるようになる．

PC/UNIX[*1] 系サーバの入出力処理機構では，ハードウェア・デバイスを制御するデバイスドライバが CPU 上で実行されるため，大量トランザクション処理におけるオペレーティングシステムのオーバヘッド（デバイスドライバが CPU 上で処理される点で，デバイスドライバもオペレーティングシステムの一部とみなすことができる）はメインフレームのそれに比べさらに大きい．PC/UNIX 系サーバの入出力処理は，メインフレームに比べて弱いといわれる要因の一つはここにある．

A シリーズでは，これらのオーバヘッドを削減しトランザクション処理のスループットを向上させるため，MCP が実行していた入出力要求の始動・終了処理，タスク・イベントの管理，およびプロセッサの割り当て処理を MCP から独立させ，専用プロセッサで実行させている．

図 9 は，大型機クラスの A シリーズで機能別入出力プロセッサを搭載したシステムとそうでない従来型システムの I/O スループットを比較したものである．横軸は 1 秒間に処理する I/O の数，縦軸は CPU の空き率である．I/O の数が多くなるとそれにしたがって CPU の空き率が低下している．これは，MCP の I/O 処理による CPU 使用量が増加するためである．従来型システムでは，I/O 数が秒あたり 1000 回を越えると CPU の空き率が急激に低下し，2000 を越えると CPU の空率が 0，つまり，もうそれ以上 I/O を発生することができない状態になっている．これは I/O 処理の多重度が上がり，排他制御やコンテキスト・スイッチによるオーバヘッドが多くなるためである．しかし，機能別入出力プロセッサを搭載したシステムでは，I/O 数が 2000 を越えても CPU の空き率はまだ 70 %近くあり，さらに I/O を発生させることが可能である．このことは，大量の入出力要求を同時処理する場合，オペレーティングシステムのオー

バヘッドがI/Oスループットに大きな影響を与えることを示している．また，機能別入出力プロセッサを搭載したシステムは，排他制御やコンテキスト・スイッチによるMCPのオーバヘッドが機能別プロセッサに分散され，CPUの利用度が大幅に向上することを示している．

とくに，秒当たり1000回を越えるI/O処理を必要とする事態は，大量トランザクション処理における処理効率向上のためにメモリをディスク代わりに利用するディスクキャッシュやメモリディスクの技術の実装によってもたらされる．すなわち，大量入出力処理には，入出力処理時間の短縮と同時に，入出力処理に要するオペレーティングシステムのオーバヘッドを大幅に削減する技術が同時に実装される必要がある．この意味において，機能別入出力プロセッサを搭載したシステムは，大量トランザクション処理により適したシステムだといえる．

2.2.3 MCPの入出力処理

つぎに，MCPおよび機能別入出力プロセッサの入出力処理における，互いの役割分担について具体的に述べる．

従来型システムのMCPは業務プログラムから入出力要求を受け取ると，IOCB(入出力制御ブロック)と呼ばれるデータ構造を主記憶上に作成する．IOCBにはその入出力要求を実行するために必要なさまざまなデータが入っている．MCPはIOCBを作成した後，その入出力要求を実行する入出力デバイスの状態を調べ，その入出力デバイスへの最適パスを選択する．もし該入出力要求がミラーディスクの読み込み要求であればミラーメンバーのなかで最も早くアクセスできるメンバーでのパスを選択する．またそれが書き込み要求であれば全ミラーメンバーに対してパスの選択を行う．その後MCPは該IOCBを適切な入出力デバイスの待ち行列に挿入し，入出力プロセッサに割り込みをかける．入出力プロセッサは渡されたIOCBをその入出力デバイスを制御する一連の制御装置に渡し，入出力動作を開始させる．

制御装置が入出力デバイスを制御し，データ転送を完了すると，入出力プロセッサはその処理結果をIOCBに入れ，IOCBをリザルトキューに挿入して，MCPに割り込みをかける．MCPは入出力終了割り込みをうけると，リザルトキューからIOCBを取り出し処理結果を調べる．もし該IOCBがミラーディスクへの書き込み終了であれば，ミラーメンバー間の書き込みの同期がとれたかどうか確認する．その後MCPは該IOCBが示しているイベントを生起させ，業務プログラムに入出力要求が終了したことを知らせる．その時，可能であればその業務プログラムにCPUの割り当てを行う．

これらの処理は，ミラーディスクに関する処理を除けば，基本的にどのオペレーティングシステムでも行っている処理であり，MCP特有の処理というものではない．たとえばPC系サーバのオペレーティングシステムであるWindows NT[*2]の入出力マネージャは，業務プログラムから入出力要求を受けると，入出力要求パケット(IRP)を作成しそのIRPをデバイスドライバに渡す．デバイスドライバは入出力要求をデバイスキューに入れるようになっている．また入出力終了時，デバイスドライバはデバイスからの割り込みを受け，データ転送を行い，入出力マネージャに知らせる．入出力マネージャはIRPを消去し，ファイルハンドルをシグナル状態にセットし，入出力要求が終了したことを知らせる．

ちなみにWindows NTは，NTファイルシステム(NTFS)にてフォールトトレランス機能を強化している．これはトランザクション処理サーバが具備すべき信頼性がいかに重要であるかを認識し，メインフレームとの比較において明らかに劣っている信頼性を強化しようとしたものである．

たとえばNTFSのFtDiskと呼ばれるディスクドライバにてミラーディスクをサポートすることになっている．しかし信頼性の強化はシステム自体を重くし，システムスループットを低下させる要因にもなりうるのである．ミラーディスクはディスクデバイスの障害に対して100％の信頼性を実現するが，ミラーメンバ間のデータの整合性を保持するための処理，たとえば書き込み同期確認や一時的に停止しているミラーメンバに対するオーディット処理などは，明らかにオーバヘッドとなる．そのハンドリングをディスクドライバで実行しようというのであるから，ミラーディスクを導入した場合，さらにCPU資源を必要とするであろう．

機能別入出力プロセッサを搭載したシステム上のMCPが実行する処理は，業務プログラムから入出力要求をうけると主記憶上にIOCBを作成するのみである．その後の処理，たとえば最適パスの選択や入出力終了処理，ミラーディスクの操作などは機能別入出力プロセッサがすべて行うので，MCPのオーバヘッドはなく，入出力処理のために要するCPU資源量を減少することができる．

表1は機能別入出力プロセッサを搭載したシステムとそうでないシステムにおける

表 1　MCPの入出力処理

従来型	機能別入出力プロセッサ搭載型
入出力開始 IOCBの作成 デバイス・テーブルのロック デバイスの状態チェックおよび更新 ミラーディスク操作確認 デバイス・テーブルのロック解除 最適パスの選択 待ち行列のロック IOCBを待ち行列に挿入 待ち行列のロック解除 入出力プロセッサに割り込みをかける	IOCBの作成 IOU（入出力ユニット-後述）にIOCBを渡す
入出力終了 入出力終了割り込み 待ち行列のロック IOCBの取り込み 待ち行列のロック解除 ミラーディスク書き込み同期確認 入出力操作に要した時間などの統計情報の積算 入出力終了イベントの生起 入出力待ちタスクの再開	なし

MCP 入出力処理の違いをまとめたものである．

2.2.4 機能別入出力プロセッサ

図 10 は A シリーズ・ハードウェアの主要モジュールの構成図である．機能別入出力プロセッサは入出力処理機構（IOM）と呼ばれるモジュールに組み込まれ，タスク制御ユニット（TCU），入出力ユニット（IOU），データ転送ユニット（DTU），チャネル管理ユニット（CMU）から構成される．機能別入出力プロセッサはメモリ・インタフェース・ユニット（MIU）を介して主記憶に直接アクセスすることができる．各プロセッサ間（CPU を含む）のコミュニケーションは，主記憶を介し 4 ワード（24 バイト）のパケットの受け渡しにより行う．主記憶を介するといっても，実際にパケットをメモリに書き込むのではなく，パケットは主記憶制御部を通り抜けるだけで目的のプロセッサに伝達される．

MCP が実行していた入出力に関する処理を，MCP に代わって実行しているのは，主にタスク制御ユニット，入出力ユニットと，データ転送ユニットである．

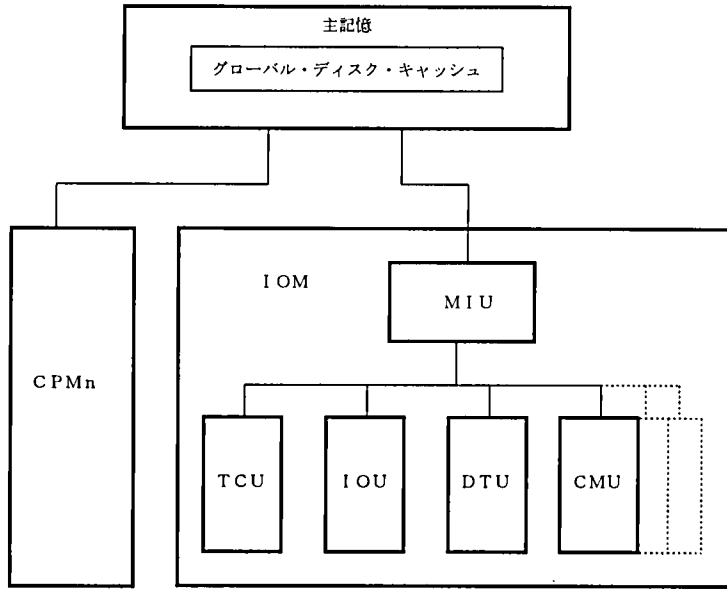


図 10　機能別入出力プロセッサ構成図

1)　TCU（タスク制御ユニット）

A シリーズで実行している業務プログラムをタスクと呼ぶ．TCU はタスクの状態，たとえば待ち，実行可能，実行中などの状態を管理し，タスクのプライオリティによるスケジューリングを行い，タスクに対してプロセッサの割り当てを行う．また，TCU はシステム内のすべてのイベントを管理する．タスクがイベントを操作するとき，TCU に対してイベントの割り当てを要求し，TCU から割り当てられたイベントのトークンをもらう．以後 MCP は，このトークンの中のイベントタスクがイベントの生起要求を行うと，TCU はそのイベントを生起状態にし，そのイベントを待っているタスクを実行可能状態にし，可能であればそのタ

スクにプロセッサを割り当てる．IOUから入出力処理の終了通知をもらった場合も同様に，そのIOCBの中にあるイベントを生起し，その入出力の終了を待っているタスクを実行可能にする．

このようにTCUの機能はオペレーティングシステムのプロセス・スイッチングそのものである．にもかかわらず，TCUがIOMに組み込まれている理由は，入出力処理の最後には必ずその終了を待っているタスクを実行可能状態にする必要があること，それに，トランザクション処理におけるタスクのコンテキスト・スイッチは多くの場合，入出力要求により発生するので，タスクやイベントを管理する機能をIOMに組み込むことは，トランザクション処理を効率良く行うためには効果的だからである．

2) IOU（入出力ユニット）

IOUはMCPから渡された入出力要求の始動とその終了処理を行う．入出力要求の始動時，指定されたデバイスの状態を調べ，最適パスを選択し，IOCBに開始時刻を入れ，そのパスを制御するCMU(チャネル管理ユニット)にIOCBを渡す．もしその入出力要求がミラーディスクの読み込みであれば，ミラーメンバのなかで最も早くアクセスできるメンバを探しその装置への最適パスを選択する．また，それがミラーディスクへの書き込み要求であれば，全ミラーメンバへの書き込み要求を始動する．さらにその入出力要求がキャッシュ指定されているディスクに対するものであれば，その入出力要求をDTU（データ転送ユニット）に渡し，データの転送を指示する．

DTU，CMUがデータ転送を終了すると，IOUはIOCBを調べその入出力がエラーなく正常に終了したかどうか確認し，その入出力の終了時刻をIOCBにいれ，TCUに入出力操作が終了したことを知らせる．その終了がミラーディスクの書き込み終了であれば，OWL（Outstanding Write List）と呼ばれるミラーメンバ間のデータの整合性を司るテーブルを保守し，全メンバへの書き込みの終了を確認する．

3) DTU（データ転送ユニット）

DTUは，グローバルディスクキャッシュと入出力バッファ間のデータ転送を行う．書き込み要求の場合，入出力バッファからグローバルディスクキャッシュに，また読み出し要求の場合，グローバルディスクキャッシュから入出力バッファにデータを転送する．データの転送は主記憶内の移動であり，0.1 ms以内で終了する．

4) CMU（チャネル管理ユニット）

CMUは，入出力デバイスを接続しているチャネルを管理する．チャネルは増設することができ，またCMU自体も増設することができる．これにより入出力デバイスの拡張性を保証すると共に，代替パスの設定による冗長性，信頼性を確保することができる．

2.2.5 入出力処理の流れ

MCPの入出力に関する処理を機能別入出力プロセッサにオフロードしたことで，MCPの入出力処理がいかに簡素化されオーバヘッドが少なくなったかを，典型的な

システムを例に，Aシリーズ入出力処理の流れを追いながらそのメカニズムを説明する．

ここに，二つのCPU（CPM 4，CPM 5）を持ったAシリーズシステムがある．このシステムでは，A，B，Cの三つのタスクが起動され，各タスクの優先度はそれぞれ，70，60，50である．今，タスクAがCPM 4で，またタスクBがCPM 5で実行中である．タスクCは実行可能状態であるがまだCPUが割り当てられておらず，CPU待ち状態にあるとする（図11(a)）．

タスクBが入出力要求を行うと，

① MCPは，タスクBの上で実行しその入出力要求をもとに入出力制御ブロック（IOCB）と呼ばれるデータ構造を主記憶上に作成する．

② タスクBのMCPは，TCUに対して，その入出力要求の終了を知らせるイベントを獲得するための要求を出す．

③ TCPはイベントEを割り当て，それを示すトークンをタスクBのMCPに返す．

④ タスクBのMCPはTCPからもらったイベント・トークンをIOCBに入れる（図11(b)）．

タスクBのMCPがIOCBを完成すると，

① IOUにその入出力を開始するよう要求する．

② IOUは要求を受け取ると，タスクBのMCPに要求受け取り確認を返す．

③ タスクBのMCPは，IOUが入出力を開始したことで，タスクBを入出力終了待ちにするため，TCPに対してタスクBをイベントE待ちにするよう通知する（図11(c)）．

TCUがタスクBの待ち通知を受けると，

① TCUは，イベントEの生起状態を調べ，まだ生起されていなければ，タスクBを待ち状態にする．

さらに，CPU待ち状態にあるタスクCにCPUをアサインするため，CPM 5にコンテキスト・スイッチを行い，タスクCを実行するよう指示する．

② CPM 5は，タスクBからタスクCにMove Stack(コンテキスト・スイッチ)を行い，TCUにタスクCを実行することを通知する（図11(d)）．

タスクBが発行した入出力要求が終了すると，

① IOUは，タスクBに入出力終了を知らせるため，IOCBに入っているイベントを生起させるようTCUに要求する．

② TCUは，イベント生起要求を受け取ると，そのイベントを生起状態にし，IOUに対してイベント生起確認を返す．

③ さらにTCUは，そのイベントを待っているタスクBを実行可能状態にし，CPU待ち行列に挿入する（図11(e)）．

タスクBが実行可能状態になると，

① CPU待ちになったタスクBの優先度が，実行中であるタスクCの優先度より高いので，TCUは，タスクCを実行しているCPM 5に対して，タスクBを実行するよう指示する．

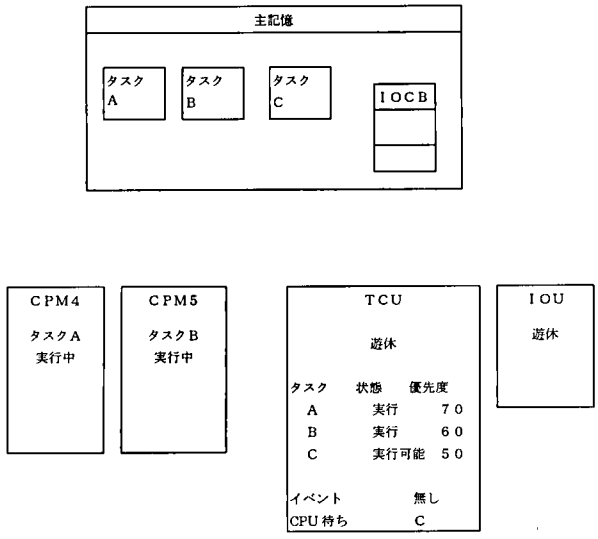


図 11(a) 入出力処理の流れ

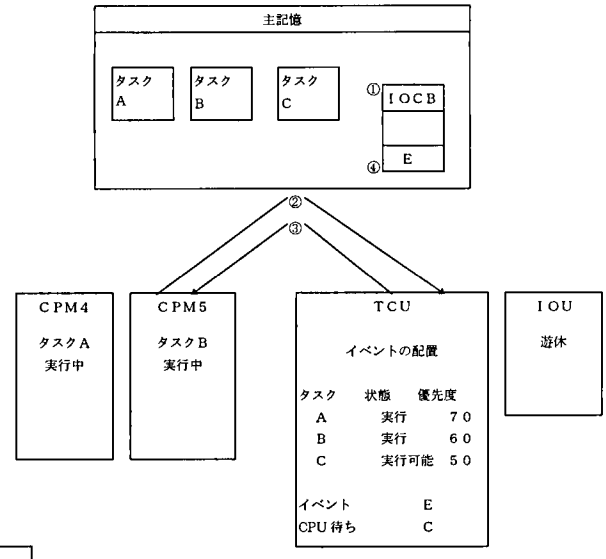


図 11(b) 入出力処理の流れ

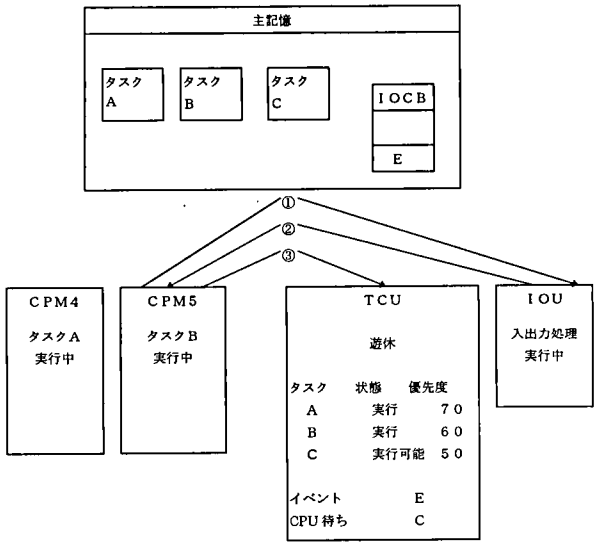


図 11(c) 入出力処理の流れ

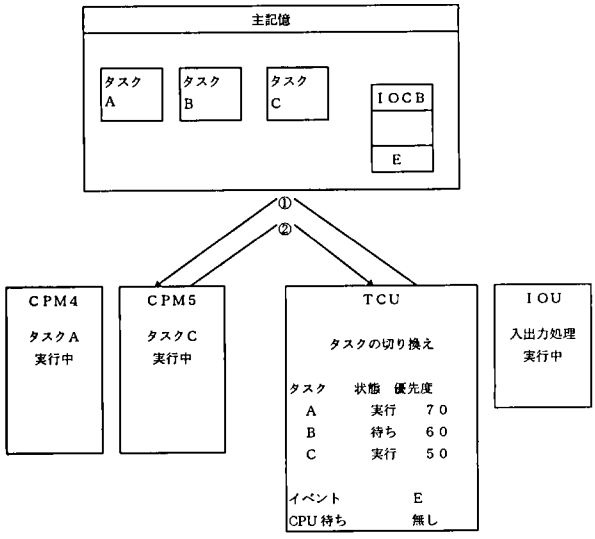


図 11(d)　入出力処理の流れ

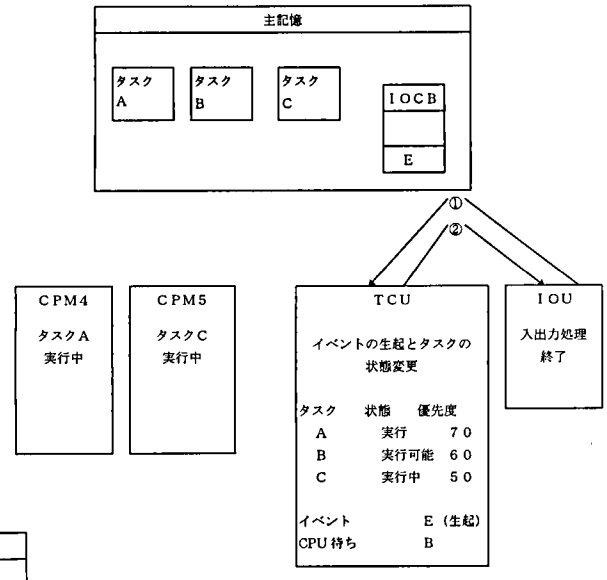


図 11(e)　入出力処理の流れ

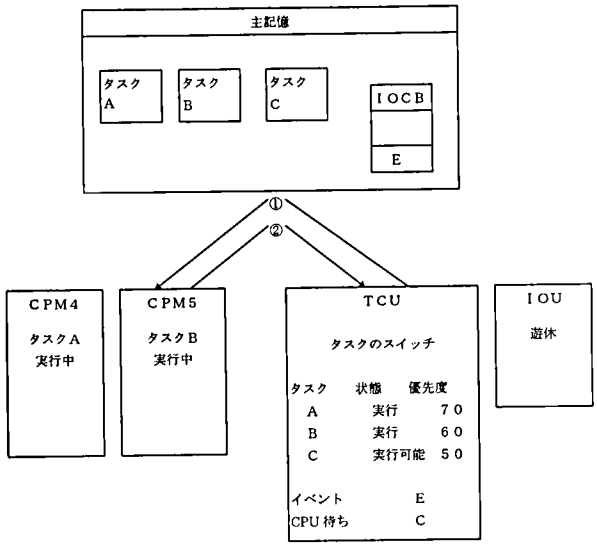


図 11(f)　入出力処理の流れ

② CPM 5 は，タスク C からタスク B に Move Stack を行い，TCU に対して，タスク B を実行したことを知らせる．これによりタスク B が実行中になり，タスク C が実行可能状態になり，CPU 待ち行列に挿入される（図 11(f)）．

3. I/O 処理技術を支えるソフトウェア技術

入出力のスループットは個々の入出力の応答時間，および並行して実行できる入出力の多重度によって決まる．入出力応答時間はオペレーティング・システム（OS）のオーバーヘッド，すなわち CPU 処理もあるが大部分はデータのアクセスおよびチャネルを介した入出力装置とのデータ転送のハードウェア処理である．したがって入出力処理時間を短くするためには，入出力プロセッサ，チャネルおよび入出力装置といったハードウェアを高性能なものを使用することが近道ではあるが，特定の装置に入出力が集中した場合の OS の制御も欠かせない．

一方，入出力の多重度を高めるためには，同時に発生する多数の入出力をそのシステムが持つハードウェア資源に如何に分配し有効利用するかというソフトウェア処理に多くを依存する．これら入出力性能を上げるためにメインフレームの OS が行っている入出力の高速化，多重化に関する処理技術の概要を以下に紹介する．特に，多数のチャネルおよび入出力装置を接続するメインフレームの拡張性を活かし入出力の多重度を高める入出力経路の操作について詳しく述べる．

また，ミッション・クリティカルな業務を扱うメインフレームでは高性能である以上にシステムの堅牢さ，およびデータの高い信頼性・可用性が要求される．これらは多年にわたり改良が加えられてきたメインフレームの OS が特に力を発揮するところである．ハードウェア障害が発生した場合の OS の回復処理は実にきめ細かで，システムの存続，データ・アクセスに向けて可能な限りの回復手段を講じる．この OS による入出力障害の回復処理についても記述する．

3.1 OS の入出力処理技術の概要

1) CPU 処理と非同期な入出力処理

プログラムは入出力要求を発行すると，その入出力が完了するまで待つことなく CPU の制御を受け，業務処理の続行あるいは別の入出力を発行することができる．すなわち，一つのプログラムから多数の入出力を並行して実行することが可能である．プログラムは入出力の完了を入出力制御バッファに返される完了ステータスの監視または入出力完了監視アクティビティが起動されることにより検出できる．

2) 入出力実行順序の最適化

OS は入出力が特定の装置に集中した場合にその実行順序を最適化する．プログラムの優先度を入出力の実行順序にも反映するとともに，追い越しが行われた入出力については順次優先度を高めて行くことにより優先度の低い入出力が一定時間以上待たされることがないよう制御する．ディスク入出力ではヘッドの移動方向を計算し，ランダム入出力の移動時間の無駄を極小化する．

3) 装置数分の入出力の並行実行

OS はシステムに接続されている全ての入出力プロセッサ，装置系列，制御装置

および装置についてそれぞれ独立した制御構造体を形成する．入出力の実行を装置構造体（UST；Unit Status Table）で管理する．装置構造体は特殊な場合を除き互いに独立して制御されるため，システムに構成されている装置の数だけの入出力を並行して実行することが可能である．

4)　入出力処理の優先

OS の入出力処理には最高の CPU 優先度が与えられる．特に入出力完了割り込み処理は，割り込みを起こした入出力プロセッサを占有するため，他の割り込みを許さない実行モードで動く．また，OS の入出力後処理が終わりプログラムに制御を戻すときにはプログラムの優先度を上げ，待ち状態にあるプログラムを優先的に制御する．

5)　トランザクション入出力処理の高速化

トランザクション入出力では入出力処理ステップ数，すなわち OS のオーバーヘッドを押さえるため，通常の入出力とは違う特別な処理を行う．OS は入出力制御のため専用のバッファ・プールを保有している．通常の入出力ではプログラムから入出力が実行される度にそのバッファ・プールから必要なバッファを獲得し，装置に適した入出力制御構造体（CP；Channel Program）を形成して入出力に使い，入出力が完了すると元のバッファ・プールに戻す．トランザクション入出力では一度形成した CP を入出力完了後にバッファ・プールに戻さず，装置の種類ごとに用意した専用のプールで保管する．同種の装置に対する次の入出力において待機中の CP を再利用することにより CP 形成処理を最小限に押さえる．

6)　データの二重化

大容量記憶域上のデータを二重化する機能である．一般的に採用されているディスク装置ごとの二重化以外に，トランザクション・データおよびデータベース更新履歴はファイル単位の二重化も組むことができる．二重化データの読み出しでは空いている装置を選択して入出力時間の短縮を図る．

7)　メモリファイル機能

メインフレームシステムが持つギガバイトメモリの一部を大容量記憶域ファイルとして使用する機能である．メモリアクセス速度でのデータアクセスを可能としている．

8)　ディスクキャッシュ機能

繰り返し参照するディスクデータは主記憶域にキャッシングできる．入出力装置に対する物理的な入出力数の削減とメモリアクセス速度の応答時間が実現される．

9)　ファイル割り当ての最適化

ファイル領域の物理ディスク装置への割り当ては，プログラムが特定装置を使用する場合を除き，OS が最適な装置選択および領域の割り当てを行う．そこではできるだけ同じディスク上の連続領域を割り当て，ファイルの分割配置に伴う処理の過負荷を極小化している．

10)　ファイル拡張処理

プログラムはディスク装置の物理容量にとらわれることなくファイルを拡張す

ることができる．ファイルが一つの装置で収まらない場合，割り当てる装置がプログラムから特に指定されていなければ，OSがファイル領域を複数の装置上に分割して割り当てる．また，データを書き込んで行くにつれて既に割り当てた領域では不足となった場合は，入出力処理内でファイルを動的に拡張する．

11) 入出力プロセッサに対する入出力の発行

OSから入出力プロセッサに対するデータ入出力の発行は主記憶域上のテーブルに入出力命令を書き込むだけで終了する．命令語を実行して入出力プロセッサに割り込みを掛けることが無いためプロセッサ間インタフェースのオーバーヘッドが押さえられる．入出力プロセッサはCPUとは非同期に主記憶上のテーブルを監視してOSから渡された入出力命令を実行する．

3.2 入出力経路選択処理

メインフレームは普通一つの入出力装置に対して複数のチャネルあるいは制御装置を通してアクセスできる装置構成を組む．入出力経路の多重化は，ある経路上のハードウェアで障害が発生した場合の代替アクセスを確保すると共に，負荷分散による入出力性能の向上に寄与する．多数ある経路を如何に使用して入出力の分散を図るかはシステムの入出力性能に直接影響する．OSは以下に示す主経路の選定，優先経路の設定，および実際の入出力経路の選定の3段階の選定処理を行い入出力で使用する経路を決めている．

まずOSはシステム初期化時に制御装置ごとにそのチャネル経路を通常の入出力で使用する主経路と，主経路で障害が発生したときに使用する代替経路に分ける．この区分けに当たってはチャネルに繋がる制御装置の数が基準となる．制御装置との経路を持つチャネルの中で，繋がっている制御装置の数が最小のチャネルを求め，その最小数の一定量(2倍)以内の制御装置を持つチャネルとの経路を主経路とする．これにより一つのチャネルに多数の装置系列を接続した場合に，そのチャネルに入出力が集中することを回避している．

次に，システム稼働中は主経路に設定した経路の中から制御装置ごとに1経路だけを優先経路に決める．優先経路は一定時間ごとにサイクリックに切り替える．優先経路を制御装置につき1経路だけとすることは，制御装置間で主経路数に差がある構成における入出力の片寄りを回避するために必要である．たとえば，図12の装置構成における主経路および優先経路を求めてみる．

装置系列Cの制御装置CUBはA2,B2,C3およびD2の各チャネルとインタフェースを持つ．チャネルA2にはCUBを含めて5台の制御装置が繋がっている．チャネルB2にはCUBの1制御装置だけ，チャネルC3には2台そしてチャネルD2には3台の制御装置が繋がっている．この構成ではCUBの繋がるチャネルの中ではチャネルB2が最小数1の制御装置を持つ．この最小数の2倍，すなわち2台までの制御装置を持つチャネルに繋がるCUBのインタフェースI2およびI3(図の太線)が制御装置CUBにおける主経路となる．この2本の主経路が交代で優先経路(図の○印経路)になる．

さらに入出力命令を発行するときに制御装置ごとに設定された優先経路の中から実際に使用する経路を決める．これには優先経路上のチャネルおよび制御装置の下で実

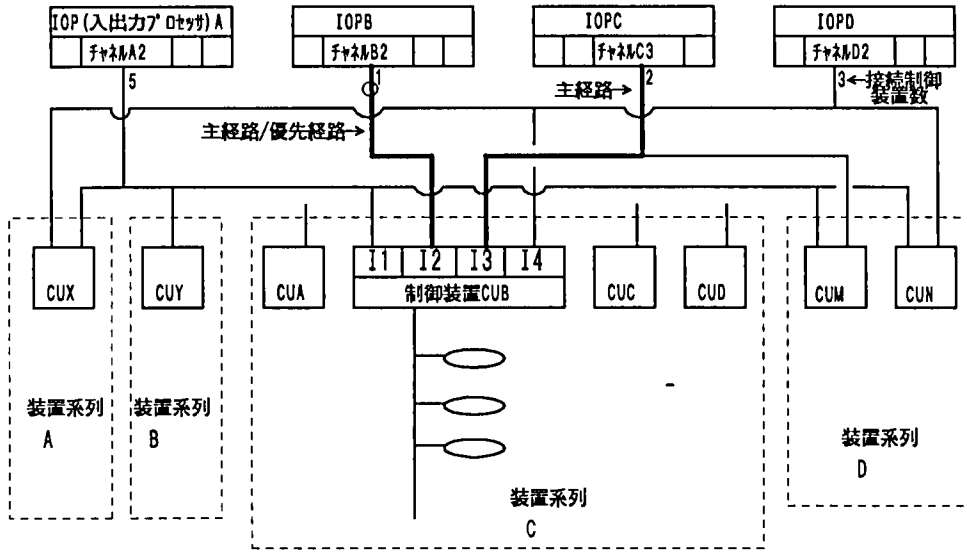


図 12　入出力主経路・優先経路の選定例

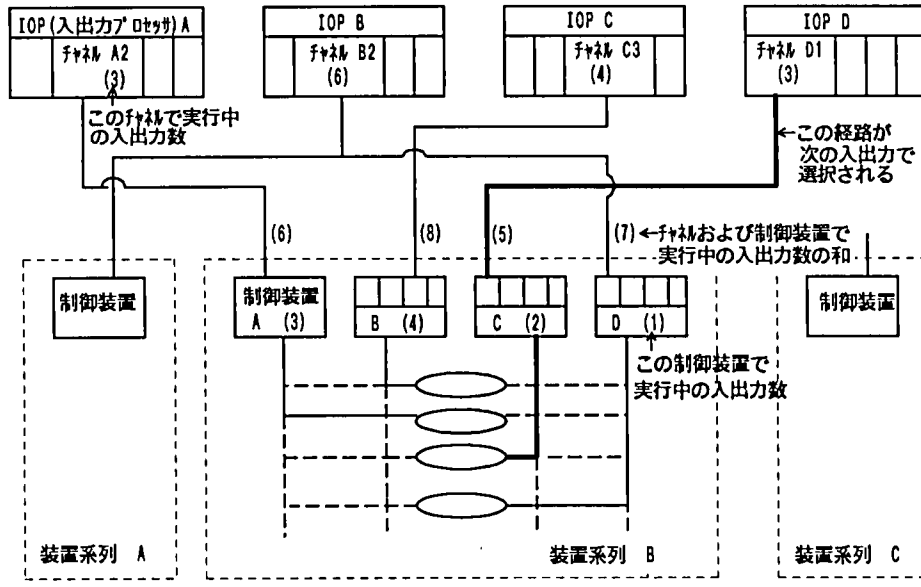


図 13　実入出力経路の選定例

行中の入出力の数を比較する．チャネルの下で実行中の入出力数と制御装置の下で実行中の入出力数の和が最も少ない経路を求めてその入出力で使用する経路とする．

図 13 の例で入出力経路を求めてみる．

装置系列 B に対して 4 本の優先経路(チャネル A 2-制御装置 A，チャネル B 2-制御装置 D，チャネル C 3-制御装置 B およびチャネル D 1-制御装置 C)がある．この中でチャネルで実行中の入出力数と制御装置で実行中の入出力数の和が最も少ないのはチ

ャネルD1（実行入出力数3）と制御装置C（同2）を通る経路である．したがってチャネルD1−制御装置Cを通る経路（図の太線）が次に実行される入出力で選ばれる．

3.3 入出力障害回復処理

入出力障害は大きく次の種類に分類される．

- 入出力プロセッサ障害
- チャネル障害
- 制御装置障害
- 装置障害
- 主記憶域障害
- 時間切れ
- 非ゼロ状況コード

上記障害の報告は入出力完了割り込み，または入出力とは独立した入出力プロセッサ起動の割り込みを通じて報告される．ただし時間切れについてはOSが入出力プロセッサに対して入出力命令を発行した後規定時間内に入出力完了割り込みが発生しない現象で，OSが判断する．これらの障害に対してOSは以下の回復処理を行う．いずれの障害回復処理も入出力の再発行によるデータ破壊の危険性がある場合を除き，装置にアクセス可能な全ての経路を用いて入出力の再発行を試みる．再発行の可否および回復不能と判断するまでの試行回数は装置の種類，構成，および障害の種類ごとに定義する．

- 入出力プロセッサの初期化/切り離し

  入出力プロセッサに対する制御系の命令が時間切れになった場合に行う回復処理で入出力プロセッサの初期化が必要である．初期化が成功した場合は入出力プロセッサの使用を継続する．初期化ができなかった場合はその入出力プロセッサをシステムから切り離して以後の入出力での使用を止める．

- 記憶域の初期化/切り離し

  主記憶域障害が発生すると，入出力プロセッサから障害の発生したメモリ番地と復旧方法（RA；Recovey Action）が報告される．OSはRAに従い障害記憶域の初期化または切り離しを行う．

- チャネルの切り離し

  チャネル障害，制御装置障害などのある種の障害では，障害の発生が装置の接続されているチャネルからの割り込みの連続発生という形でOSに伝わることがある．このとき割り込み処理によるシステムの性能低下を防ぐため，OSはそのチャネルからの割り込み停止命令を入出力プロセッサに対して発行する．特にハードウェア障害の通知を受けた場合には数回の割り込み発生を持ってチャネルの切り離しを行う．

- チャネルの初期化

  時間切れ，非ゼロ状況コード障害に対してはチャネルの初期化が必要である．OSは入出力を発行したチャネルの状態を入出力プロセッサから入手する．チャネルでの滞留または制御装置の混み状態を検出した場合はチャネル全体の初期化を行う．このときチャネルで実行中の入出力を全て一旦廃棄し，チャネル

の初期化後に再発行する．すでに入出力が装置に渡った状況，あるいはチャネルおよび制御装置は機能している状況を検出した場合は装置経路（サブチャネル）だけの初期化を行い，時間切れ・非ゼロ状況コードを受けた入出力を再発行する．

・入出力経路の切り離し

再実行可能な障害の場合，OS は先ず障害発生経路を使用して入出力を再発行する．障害発生経路での再実行を規定回数行っても入出力が成功しないときは他の経路を使用して回復処理を続ける．代替経路からの入出力が成功した場合は経路特有の障害と考えられるため以降の入出力では障害経路の使用を止める．しかし障害経路上の入出力プロセッサ，チャネルおよび制御装置はそれらを使用する他の入出力経路のために残す．全ての経路を通しても成功しない場合は特定経路の障害とは考えられず，経路の切り離しは行わない．

・入出力データの再編成

ディスク装置におけるデータ・エラーのように単に入出力命令を再発行するだけでなく，レコードの境界から入出力を開始することにより回復が可能となる障害もある．この種の障害では OS は障害の発生したレコードを検出し，残りのデータを障害レコードの境界より転送するよう入出力制御構造体を編成し直して回復を試みる．

障害の種類とそれに対して行われる回復処理の対応を表 2（○印）に示す．

表 2　入出力障害の種類と回復処理の対応表

回復処理 / 障害内容	IOP 初期化/切り離し	ﾒﾓﾘｰ初期化/切り離し	ﾁｬﾈﾙ切り離し	ﾁｬﾈﾙ/経路初期化	入出力経路切り離し	CP 再編成/再発行
IOP 障害	○		○		○	○
ﾒﾓﾘｰ障害		○			○	○
ﾁｬﾈﾙ障害			○	○	○	○
制御装置障害			○	○	○	○
装置障害						○
時間切れ				○	○	○
非ｾﾞﾛ状況ｺｰﾄﾞ				○	○	○

1. メインフレームでサポートするチャネルの種類

今日の情報・通信技術の進歩は周知の通り著しいものがある．この技術革新の時代に情報化産業でリーダーシップを取るためには最新の技術を採用しかつ提供することが必要である．一方でこれまで長い年月を掛けて蓄積して来た技術・資産を活かすことも重要である．多年にわたり使われ続けて来たメインフレームにおいてはこれが特に要求される．メインフレームの入出力チャネル分野では最先端技術を採用しつつ従来の入出力機器の接続もサポートすることでこの課題に応える．以下にシリーズ 2200 が現在サポートしているチャネルおよび近い将来導入する予定の AFC（ANSI Fiber Channel）チャネルおよび ATM（Asynchronous Transfer Mode）チャネルについてその概要を記述する．

1) ブロック多重チャネル

導線ケーブルにより接続するチャネルで8ビット幅の転送を行う．インターロック機構で3.0メガバイト/秒，ストリーミング機構では4.5メガバイト/秒のデータ転送速度を持つ．FIPS-60プロトコルに準拠し，1970年代より長期間にわたり使われ続けているチャネルである．

2) SCSIチャネル

ANSI X 3. 131-1986 SCSI標準およびANSI X 3. T 9. 2/85-52で定義される標準コマンドに準拠し8ビット幅で10メガバイト/秒または16ビット幅(Wide SCSI)で20メガバイト/秒の転送速度を持つチャネルである．SBC(SCSI Bus Controller) を搭載するディスク装置のサポート，マルチ・イニシエータを可能とするなど拡張性もあり，シリーズ2200では中下位機種における主入出力装置に位置付けられている．

3) Ethernet[*3]チャネル

Ethernet IEEE 802.3 LANを接続するチャネルで2メガバイト/秒のデータ転送速度を持つ．従来LAN接続はブロック多重チャネルにアダプタ装置を接続してチャネルのパラレル・インタフェース信号とLANのシリアル・インタフェース信号の切り替えを行っていたが，Ethernetチャネルを使用することによりLANケーブルをチャネルに直接接続できる．

4) FDDIチャネル

転送速度100メガバイト/秒のFDDIループ型LAN (ISO 9314) を接続するチャネルで，チャネルにおけるデータ転送速度も10メガバイト/秒と高速である．

5) SBCONチャネル

光ファイバ・ケーブルを接続し17メガバイト/秒のデータ転送速度を持つ高速チャネルである．個々の入出力に装置アドレスを付加するインタフェースを持ち，入出力ごとに接続装置を切り替える，装置の動的な接続・切り離し動作が可能である．SBCONディレクタと呼ばれるスイッチング・ハブがこの動的接続・切り離しを司る．SBCONディレクタを使用することにより，一つのチャネルの下に多数の入出力装置を接続したり，一つの入出力装置を複数のシステムあるいはチャネルからアクセスするなど柔軟なシステム構成が可能となる．また，SBCONチャネルは伝送媒体に光ケーブルを使うため，数10キロメートル離れた遠隔地に入出力装置を設置して使用することも可能である．

6) AFCチャネル

ANSI Fiber Channel Physical Interface (X 3.230), ANSI Fiber Channel Arbitrated Loop (X 3 T 11) に準拠し，100メガバイト/秒のデータ転送速度を持つ高速光チャネルである．入出力装置とは一対一接続，126装置群を結ぶループ接続あるいは一体nの切り替え接続が可能で多彩な構成を組むことができる．将来の主力チャネルになるものと予測される．

7) ATMチャネル

ATM LANケーブルを接続するATMチャネルの導入も計画されている．ATMはLAN/WAN共通に利用可能なこと，柔軟な帯域割付，優先制御機能を

持ち音声や映像などのマルチメディア・データ伝送に適していること，155メガビット/秒，622メガビット/秒あるいはそれ以上の高速転送を単一プロトコルで利用できること，さらにアーキテクチャとして非常に高い信頼性を有していることから次世代インフラとして最も有望視されているネットワーク技術である．このATM LANをチャネルに直接接続することによりメインフレームをシームレスなネットワーク構成に組み入れることが可能となる．

＊1 UNIXは，X/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
＊2 Windows NTはMicrosoft社の商標である．
＊3 Ethernetは Xerox社の登録商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

参考文献
［1］ Helen Custer著，鈴木慎司監修，福崎俊博訳，INSIZE WINDOWS NT，アスキ出版局，1993.4.
［2］ HELEN CUSTER著，小畑喜一/五十嵐宰監修，大西照代訳，WINDOWS NT，ファイルシステム，アスキー出版局，1995.2.
［3］ 出揃った汎用並列機とディスク/アレイ　日経ウオッチャーIBM版，日経BP社，1995.

執筆者紹介

白　崎　　宏（Hiroshi Shirasaki）
1980年岩手大学工学部情報工学科卒業．同年日本ユニシス(株)入社．Aシリーズ基本ソフトウェアの開発保守業務に従事する．現在，システムプロダクト部システム支援ソフトウェア開発室に所属．

新　津　昭　義（Akiyoshi Niitsu）
1970年早稲田大学商学部卒業．同年日本ユニシス入社．シリーズ2200のオペレーティング・システム（EXEC）の保守を中心に，2200/900，XPCなどの導入作業を実施．現在システムプロダクト部2200ソフトウェア開発室に所属．

坂　本　徳　明（Noriaki Sakamoto）
1973年北九州工業高等専門学校卒業．同年日本ユニシス(株)入社．ハードウェア・エンジニアとして客先サービスに従事した後，小型機オペレーティング・システムの保守業務に従事する．1989年より，システムプロダクト部Aシリーズソフトウェア開発室に所属．

金 井 秀 量 (Hidekazu Kanai)
1976年東京工業大学理学部卒業．同年日本ユニシス(株)入社．シリーズ2200のオペレーティング・システム(EXEC)の導入および保守業務に従事する．現在システムプロダクト部2200ソフトウェア開発室に所属．

並列システムにおける負荷平準化機能

Workload Balancing in Parallel Processing System

畑　邦明

要　約　メインフレームによる並列処理システムでは，その性能をすべて引き出すためには，クラスタ間の負荷を平準化制御することが必要である．平準化制御の対象となるシステム・リソースは，CPU利用率，I/O利用率，メモリ利用率などである．並列トランザクション処理システムでは，CPU利用率を平準化制御する必要がある．またバッチの並列処理システムでは，CPU利用率・I/O利用率・メモリ利用率を平準化制御する必要がある．

本稿は，まず平準化制御を行ういくつかの方法を解説し，次に並列トランザクション処理システムで負荷平準化制御を行うバランスマネージャの仕組みについて説明する．最後にバッチの並列処理とその平準化制御について考察する．

Abstract　In order to maximize the performance of a mainframe-based parallel-processing system, it is essential to balance the workload between clusters. System resources to be balanced include CPU, I/O and memory usages. For parallel transaction-processing systems, it is required to be balanced CPU usage ratio between clusters. For batch parallel-processing systems, CPU, I/O and memory usages ratio should be balanced.

This technical paper mainly addresses some workload-balancing methods, then describes how a balance manager balances workload in a parallel transaction-processing system, and finally considers workload balancing for a batch parallel-processing system is evaluated.

1. はじめに

メインフレームのCPUは，高速・高価なECL（エミッタ結合理論素子）技術から，より低価格なCMOS（相補性型金属酸化膜半導体）技術へと移り変わっている．これはUNIX*1系サーバとの価格競争力を回復する目的と，CMOS製CPUの処理能力が向上してきたことがその理由と言える．しかしECL製CPUの処理能力と比べると，まだその処理能力を超えるまでに至っていない．マルチプロセッサでは対応できないような，より大きなCPU能力を必要とするシステムのために，複数のメインフレームを組み合わせた汎用並列処理システムが必要となってきた．

当社では早くからメインフレームによる並列処理システムの開発に取り組んできた．そして世界に先駆けて1989年から，並列トランザクション処理システムによるエアラインの座席予約システムを稼働させている．この他にもTRITONシステム(金融機関向けの次世代勘定系システム・パッケージ)，電力システムでの稼働実績を持っている．

当社の並列トランザクション処理システムは，拡張トランザクション処理アーキテクチャXTPA (eXtended Transaction Processing Architecture) にもとづいている．これは，8台までのシリーズ2200を疎結合（Loosely coupling：疎結合システムは，複数のシリーズ2200をクラスタ間制御装置を介して結合したシステムである．ディス

ク装置も各クラスタ間で共有する．これを共用ディスクという.)することにより，データ共有型の並列処理システムを構築し，全体をあたかも一つのシステムとして稼働することを可能とする技術である．並列処理システム化によって，トランザクション処理性能（スループット）の向上と，無停止連続稼働が可能となる．

並列処理システムでは，その性能を引き出すため，クラスタ間の高速排他制御，共用データベースの高速入出力，およびクラスタ間の負荷平準化が不可欠である．前二者については拡張データ処理装置 XPC (eXtended Processing Complex) により実現されている．本稿では，シリーズ 2200 の並列処理システムにおけるクラスタ間の負荷平準化について解説する．

2. シリーズ 2200 の並列処理

シリーズ 2200 の並列処理は，大規模なトランザクション処理システムを構築する目的で開発された．並列処理システムの構成はデータベースの配置によって，データ共有型とデータ分散型（シェアード・ナッシング型）に分類する事ができる．一般的にメインフレームによる並列処理システムではデータ共有型構成，クラスタ数が数十個を超えるような超並列機ではデータ分散型を採用している．シリーズ 2200 の並列処理は，データ共有型を採用している．

シリーズ 2200 の並列処理システムでは，クラスタ間結合装置として XPC または RLP（Record Lock Processor：共用データベース・レコーダの排他制御を行う専用プロセッサ）を使用する．XPC はデータベースの更新を効率よく行うための「レコードロック制御機能」と，データ・アクセスを高速化するための「グローバル・キャッシュ機能」を持っている（図 1)．

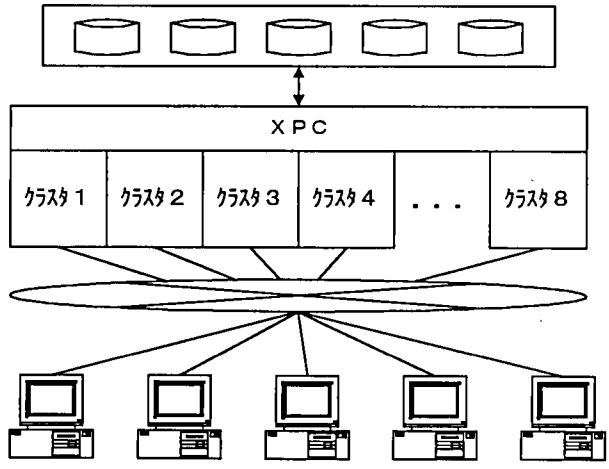


図 1　並列処理構成

レコードロック制御機能は，複数のクラスタから共通のデータベース・レコードの排他制御を行う機能で，排他制御のための負荷を最小限にすることができる．XPC のレコードロック制御機能を RLP と比較すると，ロック処理能力とロック保持容量，および耐障害性能が向上している．これは，XPC 内の複数のロック・エンジンによるロ

ック処理の並列化，XPC の大容量キャッシュ・メモリにロック・リストを作成することによる大容量化，および複数のロック・エンジン間での相互監視とバックアップ機能のためである．

グローバル・キャッシュ機能は，すべてのディスクとの入出力を高速化するもので，とくに繰り返しアクセスするデータには有効である．この結果ホットスポット（特定のデータベース・レコードにアクセスが集中する現象．この結果，そのレコードが存在するディスクのアクセス時間が長くなり効率が著しく低下する．）が発生した場合でも，データは XPC のキャッシュ・メモリからアクセスできるので，効率低下を最小限に防ぐことができる．

2.1 トランザクションの並列処理

トランザクションを並列処理するためには，アプリケーション・プログラムが参照するデータベースを共用系のディスクに配置することと，OS のシステム生成でそのトランザクションが所属するアプリケーション・グループの属性を「共用」と指定することが必要である．並列処理を行うために，アプリケーション・プログラムの変更は一切必要ない．

次に，オンライン端末とクラスタとの接続を決める．この時に考慮するのは，クラスタの障害や回線制御装置などの障害を想定して，万一それらの障害が発生した場合でも，支店・営業所・窓口などの端末がすべて使用できなくなることがないようにすることである．このためには，端末と接続先のクラスタと接続経路を，たとえば図 2 のような 2 クラスタ構成では支店内の半数の端末はクラスタ A へ，残りの半数はクラスタ B へ接続すればよい．このように接続することによって，万一クラスタ A で障害が発生した場合でも，支店内の半数の端末は正常に使用し続けることができる．障害が発生したクラスタに接続していた端末は，障害が回復するのを待って再度接続するか，正常なクラスタへ接続を切り替えてオンライン業務を再開する．

図 2 の例はクラスタ A・B 共に半数ずつの端末を受け持つことになるので，トランザクションの発生率が端末ごとに差がなければクラスタ間での負荷は等しくなる．これを並列システムにおける静的な負荷平準化と呼ぶ．しかし実際にはトランザクショ

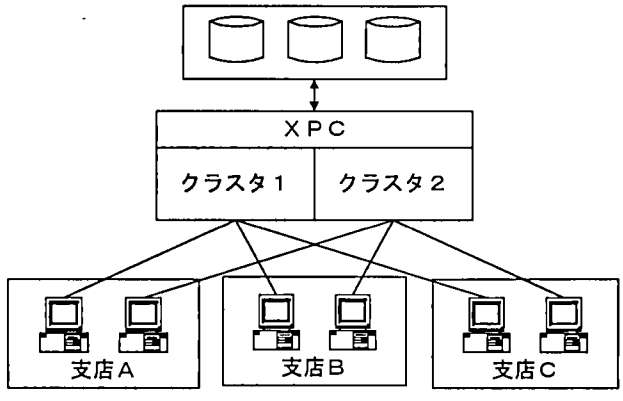


図 2　2 クラスタの並列処理システム

ンの発生率に偏りがあったり，クラスタ内で固有の処理等があって，クラスタ間の負荷が平準化しないことがある．

クラスタ間の負荷が不均衡になっていると，システム全体の処理量が増加してきた場合，あるいはピーク時に並列処理システム全体の能力が発揮できないという問題が生じる．つまり負荷の大きい（処理能力の余裕がない）クラスタのスループットが悪化してしまい，処理待ちのキューが発生するからである．この問題を解決するためには負荷（トランザクション発生率）の変動やクラスタ間の負荷量をモニタリングし，より負荷量の小さいクラスタが優先的にトランザクションの処理を行うような仕組みが求められる．

2.1.1 負荷平準化制御

先に述べたように，並列処理システムでは，処理能力を効果的に引き出すためには，クラスタ間の処理量は均衡している必要がある．クラスタ間の処理量を均衡にする制御を，ここでは負荷平準化制御と呼ぶことにする．

負荷平準化制御で対象とするシステム資源としては，トランザクション処理システムでは，CPU 利用率を平準化することが重要である．また，バッチ処理システムでは，CPU 利用率・メモリ利用率・I/O 利用率を平準化する必要がある．次にその理由を述べる．

トランザクション処理システムの場合，一般的に一つの処理単位(メッセージ処理)はバッチ処理と比べると小さい．また，一つの処理単位が消費するシステム資源の量もバッチと比較すると少ない．このため平準化制御は，負荷の変化を予測して制御を行うよりも，フィードバックの要素を大きくとった制御を行うことが求められる．つまり１秒間に数十件～数百件のトランザクションを処理するシステムでは負荷の変化を正確に予測することはできない．たとえできたとしても，そのための負荷は膨大になってしまうのは明らかである．

トランザクション処理システムでは，平準化の管理対象とするシステム資源は，CPU 利用率だけを管理すればよい．その理由は，トランザクション処理を行う大部分のプログラムは，メモリに常駐している場合が多いこと．また，データベースの入出力は，どのクラスタからでもできるので，I/O 利用率は並列処理システム全体で見ることができるからである．トランザクション・プログラムのプロファイルはほとんど似かよっており，バッチのように極端に CPU を使ったり，極端に I/O を行うようなプログラムではないこと，などがその理由である．

一方バッチ処理の場合は，フィードバックよりも負荷の予測を重視しなければならない．なぜなら一般的にバッチの負荷は大きいため，フィードバック制御では負荷の変化が大きくなってしまうためである．平準化の管理対象とするシステム資源としては，CPU 利用率，メモリ利用率，I/O 利用率などを総合的に見る必要がある．

次に，並列トランザクション処理システムでの負荷平準化を行うための技術について述べる．ただしここで述べる方式すべてが実現しているわけではない．

1) DCP（回線制御装置）

DCP を使った負荷平準化では，端末から入力されたトランザクションを負荷の少ないクラスタに振り分けて処理させる．オンライン端末と並列処理クラスタ間

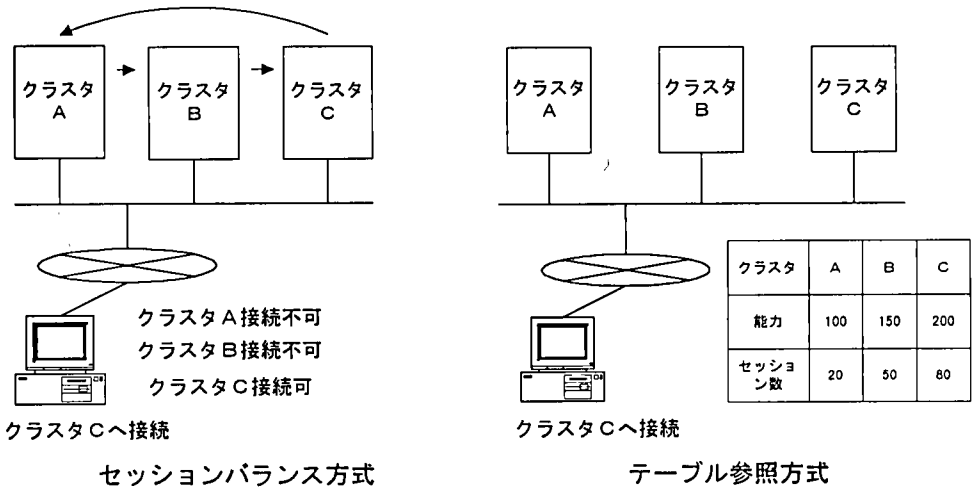


図 3　セッション・バランス方式とテーブル参照方式

のセッションを確立する際に，クラスタの能力に従ってあらかじめ決められた比率に従って，クラスタを決定するセッション・バランス方式と，各クラスタの負荷状況を表したテーブルを参照しながら接続するクラスタを決めるテーブル参照方式がある（図 3）．テーブル参照方式では，各クラスタは定期的に負荷状況を DCP へ知らせる．

DCP を使ったセッション・バランス方式とテーブル参照方式では，クラスタに負担をかけないメリットがある反面，セッション単位でしかバランス制御ができないという欠点がある．つまり一度セッションが確立してしまうと，次にセッションを再確立するまでの間は，クラスタの負荷状況に関わりなく決まったクラスタで処理することになってしまうためである．

2)　FEP（フロントエンド・プロセッサ）

並列処理クラスタとネットワークの間に接続された FEP が各クラスタの負荷の監視を行い，負荷の少ないクラスタへトランザクションを振り分ける方式である．各クラスタは定期的にクラスタの負荷状況を FEP に知らせる．FEP 方式では，DCP 方式と同様にクラスタに負荷をかけないメリットがあり，さらにトランザクション単位に負荷平準化制御ができる．しかし FEP を導入するためのコスト的な負担が必要となる．FEP 方式はトランザクション量がかなり多いシステムで採用されている（図 4）．

3)　XPC（ホスト間結合装置）

XPC を使った負荷平準化には，グローバル・キューイング方式とダイナミック・キューイング方式がある（図 5）．

グローバル・キューイング方式は，並列処理クラスタへ入力されたすべてのトランザクションを XPC のグローバル・キャッシュ領域へキューし，各クラスタはこのキューからトランザクションを拾い上げて処理を行う．

ダイナミック・キューイング方式では，並列処理クラスタへ入力されたトラン

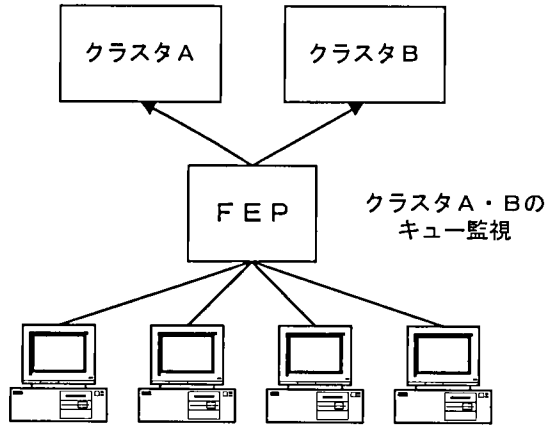


図 4　FEP 方式

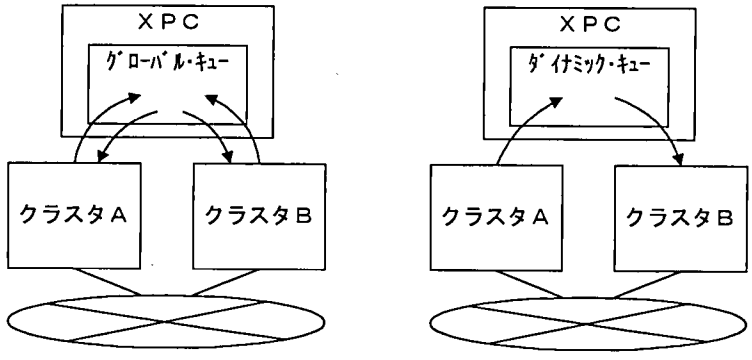


図 5　グローバル・キューイング方式とダイナミック・キューイング方式

ザクションの中で，負荷平準化のために必要なだけのトランザクションを，XPC のグローバル・キャッシュ領域へキューして，負荷平準化の対象のホストで処理を行わせる．

グローバル・キューイング方式は単純な仕組みで負荷平準化制御ができるが，平準化のための負荷が高くなる．つまり並列処理システムではもともと静的な負荷平準化が行われている（図 2）．したがって大部分のトランザクションは入力されたクラスタでそのまま処理しても良いはずである．このためグローバル・インプット・キューを経由するという負荷が，すべてのトランザクション処理に余分にかかることになる．

ダイナミック・キューイング方式では，グローバル・キューイング方式と比べ負荷平準化に必要な数のトランザクションだけを XPC にキューするので平準化のための負荷はグローバル・キューイング方式よりも軽くすることができる．

4)　アプリケーション・プログラム

クラスタ内で発生するトランザクション処理，たとえば銀行業務でのセンターカット処理などは，アプリケーション・プログラム自身，またはミドルソフトウ

ェアで負荷平準化を行う（図6）.

端末から入力されたトランザクションによって起動されるアプリケーション・プログラム自身で平準化制御を行うのは，平準化のための負荷が大きすぎるため行うべきではない.

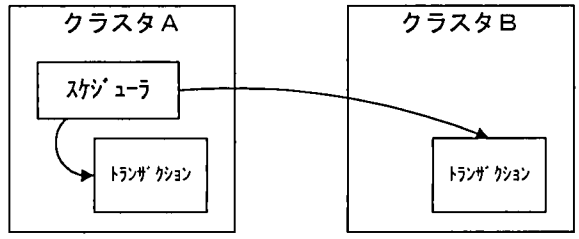


図 6　アプリケーション・プログラム方式

次項では，ダイナミック・キューイング方式を採用して並列トランザクション処理システムの負荷平準化制御を行うトランザクション・ロードバランス機能（バランスマネージャ）を紹介する.

2.1.2　バランスマネージャ

トランザクションの負荷平準化制御は，バランスマネージャによって実現する．バランスマネージャは，各クラスタ上で常駐ランとしてOS・MCB（Message Control Bank）と協調して動く．各クラスタ上で実行するバランスマネージャは，XPCを介して通信を行う.

各クラスタで実行するバランスマネージャは，それぞれ同一の機能を持つ．特定のクラスタで実行するバランスマネージャが，他のクラスタで実行するバランスマネージャを制御する方法は，並列の原則をはみ出すので採用していない.

バランスマネージャは，バランスマネージャ本体とユーティリティ・プログラムおよび，コモンバンクで構成する．本体はバランサ，モニタ，INQマネージャ，OUTQマネージャおよびXPC-MRFマネージャの五つのモジュールで構成する（表1）.

表1　バランスマネージャの構成

ユーティリティ	バランスマネージャが使用するファイルの作成と，バランス対象の端末とトランザクション・プログラムの設定を行う.
コモンバンク	メッセージ制御バンク(MCB)とバランスマネージャ本体の間のインタフェースを持つ.
バランサ	バランスマネージャの各モジュールの制御を行う.
モニタ	定期的にクラスタごとに負荷状況を得て，負荷平準化に必要なトランザクション件数を計算する.
INQマネージャ	モニタが計算した件数のトランザクションを対象となったクラスタへ振り分ける.
OUTQマネージャ	振り分けられたトランザクションからの出力メッセージをもとのクラスタへ戻す.
XPC-MRFマネージャ	XPCを介してクラスタ間の通信を行う.

バランスマネージャの構造を図7に示す.

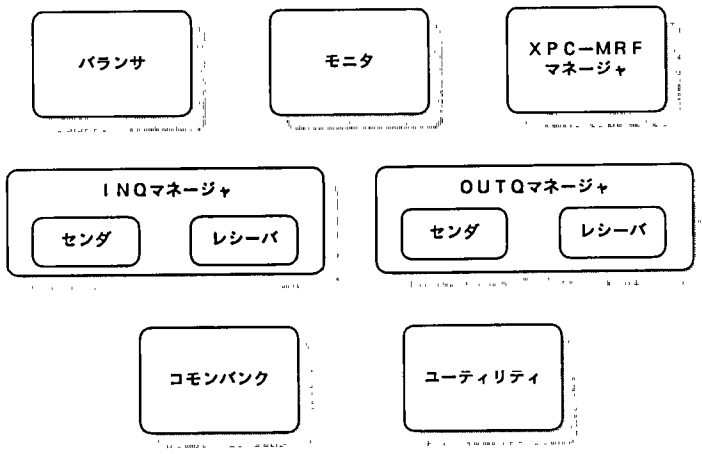


図 7　バランスマネージャの構造

1)　ユーティリティ

バランスマネージャが使う「管理ファイル」と「キューファイル」の作成，およびバランス対象とするオンライン端末とトランザクション・プログラムの設定を行う．ユーティリティは，バランスマネージャの実行に先立って実行する．これらのファイルは共用ディスク上に作成し，XPC のキャッシュ・メモリ上に常駐する．管理ファイルとキュー・ファイルの役割は次の通りである．

●管理ファイル：各ホストの CPU 利用率，バランス対象端末の番号・プログラムの一覧表，バランスマネージャの制御情報を格納する．

●キュー・ファイル：クラスタごとに 1 ファイルずつ作る．インプット・メッセージとキュー・エントリ，アウトプット・メッセージとキュー・エントリを格納する．

2)　コモンバンク

バランスマネージャのコモンバンクは MCB から呼び出され，入力されたメッセージをどのクラスタで処理すべきかをモニタの指示に従って決定する．

3)　バランサ

バランサはバランスマネージャ全体の制御と，バランスマネージャに対するキーインの処理を行う．

4)　モニタ

モニタはクラスタの CPU 利用率を定期的（1 秒ごと）に OS から入手し，管理ファイル上の自分のクラスタの CPU 利用率を更新する．管理ファイルから各クラスタの CPU 利用率を読み込んで比較する．この結果と直前の 1 秒間でバランスした件数をもとに，次のインターバルの間でどのクラスタへ何件のトランザクションを回したらよいかを計算してコモンバンクへ知らせる．

5)　INQ マネージャ

INQ マネージャは，センダとレシーバの二つの機能に分けることができる．センダは端末から入力メッセージを受け取ってキュー・ファイルへ書き込む．レシ

ーバはキュー・ファイルからメッセージを拾い上げてトランザクション・プログラムを実行する．

6) OUTQ マネージャ

OUTQ マネージャも，センダとレシーバの二つの機能に分かれている．センダはバランスされたトランザクション・プログラムからの出力メッセージを受け取ってキュー・ファイルに書き込む．レシーバは，キュー・ファイルから OUTQ エントリを拾い上げ，メッセージを読み込んで MCB へ渡し端末へ出力する．

7) XPC-MRF マネージャ

XPC-MRF マネージャは，INQ マネージャのセンダとレシーバ間，および OUTQ マネージャのセンダとレシーバ間の通信を行う．

端末から入力されたメッセージがバランスマネージャによって負荷の軽いクラスタへ渡ってトランザクション・プログラムが実行する仕組みを解説する（図 8）．

① CMS（Communications Management System）はオンライン端末から入力されたメッセージを受け取って MCB-MRF（MCB Message Retention File）へ格納する．

② MCB は OS へトランザクションプログラムのスケジュール要求を行う前に，バランスマネージャ・コモンバンクを呼び出す．バランスマネージャ・コモンバンクはこのトランザクションを自分のクラスタで実行すべきかを判断す

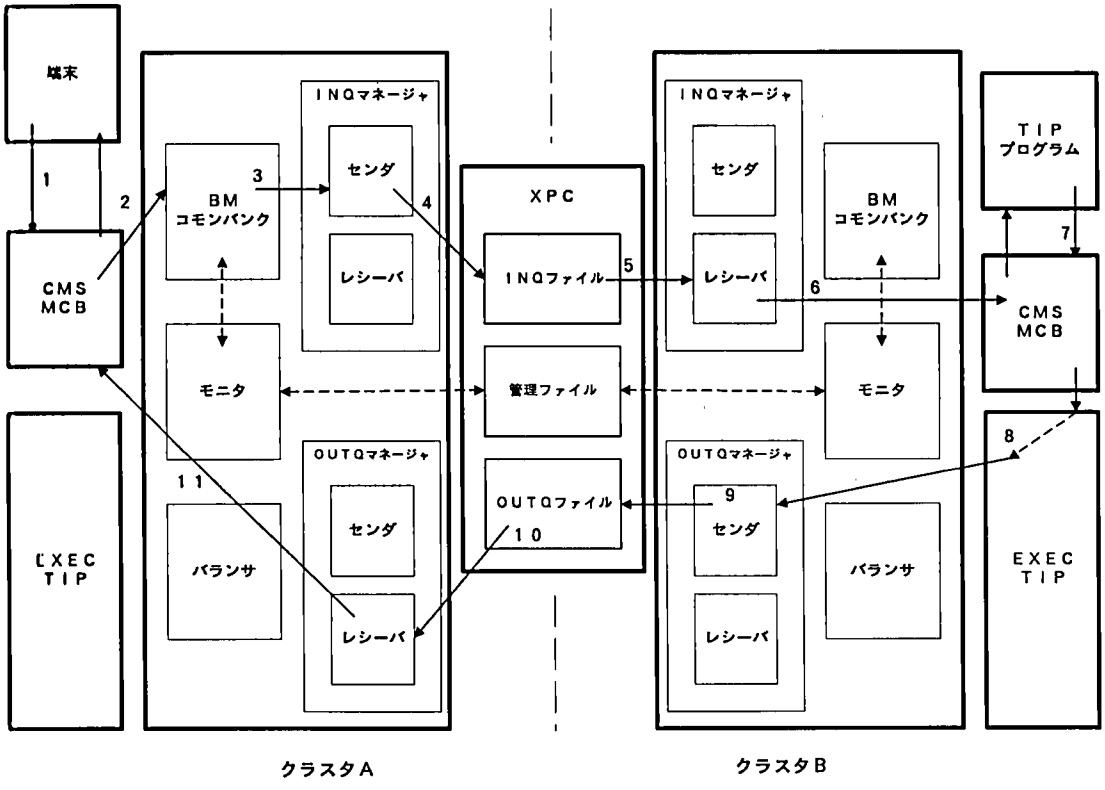


図 8　バランスマネージャの仕組み

る．バランスマネージャ・コモンバンクが自分のクラスタで処理すると判断した場合は，MCB に戻ってスケジュール要求を行う．

③ バランスマネージャ・コモンバンクで他のクラスタで実行すべきと判断すると，バランスすべきメッセージが到着したことをバランスマネージャの INQ センダへ知らせる．この場合 MCB はスケジュール要求は行わない．

④ INQ センダは入力メッセージを MCB-MRF から読み出して XPC 上のキュー・ファイルに INQ エントリを作成し，入力メッセージを書き込む．この後入力メッセージを処理するクラスタで実行している INQ レシーバへ通知する．

⑤ INQ レシーバは入力メッセージを XPC 上のキュー・ファイルから読み込んで，MCB-MRF へ格納する．この時 INQ マネージャは，①の CMS のように振る舞う．

⑥ MCB はトランザクション・プログラムのスケジュール要求を OS へ行う．

⑦ トランザクション・プログラムは入力メッセージの受取処理を行う．トランザクション・プログラムが作成した出力メッセージは MCB-MRF に格納される．

⑧ バランスされたトランザクション・プログラムからの出力は，プログラム終了時（コミット処理時）に OS からバランスマネージャの OUTQ センダへ知らされる．

⑨ OUTQ センダは出力メッセージを MCB-MRF から読み出して XPC 上のキュー・ファイルに OUTQ エントリを作成し，出力メッセージを書き込む．このあとメッセージを出力するクラスタで実行している OUTQ レシーバへ通知する．

⑩ OUTQ レシーバは出力メッセージを XPC 上のキュー・ファイルから読み込んで MCB へ渡す．この時 OUTQ マネージャは，⑦のトランザクション・プログラムのように振る舞う．

⑪ MCB へ渡った出力メッセージは CMS を経由して端末へ出力する．

バランスマネージャ実行の開始と終了は，トランザクション処理を行っている任意の時点で行うことができる．実行中は常にバランス制御を行っているわけではない．各クラスタとも予め与えておいた CPU 利用率の下限値に達していないとき，あるいは各クラスタともに CPU 利用率の上限値を超えてしまっている場合は，バランス制御を行わない．また，クラスタ間の CPU 利用率の差が許容値以内に収まっている場合もバランス制御を行わない．CPU 利用率の上限値/下限値/許容値は，自由に設定できる．

次に，バランスマネージャのモニタに採用したバランス・ロジックについて解説する．

モニタは 1 秒ごとに起動して，CPU 利用率を OS から入手し，バランスマネージャの管理ファイル上のクラスタごとの CPU 利用率を更新する．モニタは管理ファイルを読み込むことで各クラスタの直前の CPU 利用率を知ることができる．

バランスマネージャが起動した時刻 t0 でのクラスタ A と B の CPU 利用率が図 9 のようになっている場合，時刻 t1 までの間に負荷が高いクラスタ A に入力されたト

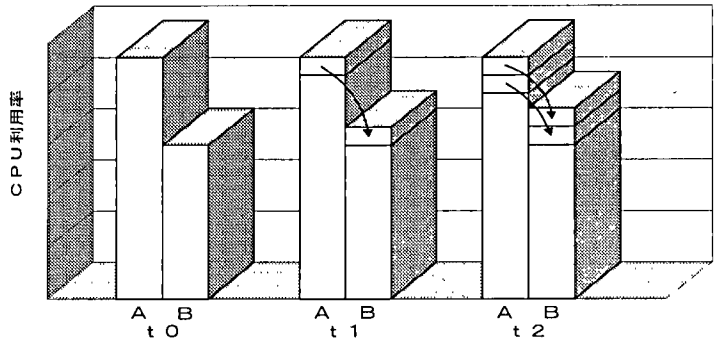


図 9　バランス制御

ランザクションから1件を負荷が低いクラスタBへ回す．時刻t1でもまだクラスタBの負荷が低ければ，次の時刻t2までの間に1件加えて2件を回す．このようにして負荷が平準化するまで回すトランザクションの件数を増やしていく．この方式では負荷が平準化するまでに時間がかかるが，t0/t1/t2のインターバルを小さくすることで対応できる．ある時点でクラスタAとBの負荷が逆転した場合，クラスタBからもAへ回したくなるが，これではトランザクションの回し合いとなってしまう．この場合はクラスタAからBへ回す件数を減らしていく．

上で述べたバランス・ロジックを簡単なシミュレーションを行って検証する．シミュレーションは3クラスタ構成をモデルとして行う．バランス制御を行わなかった場合の基礎データとして各クラスタのCPU利用率のモデルを作成した（図10）．

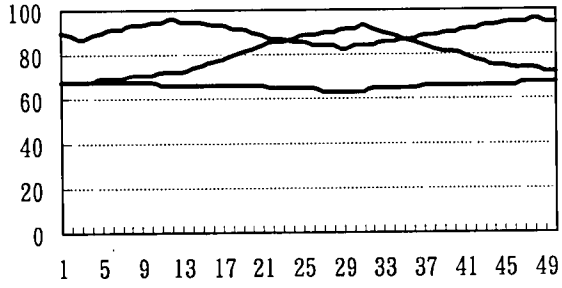


図 10　CPU利用率のモデル

- 1トランザクションのCPU利用率を1.5%に固定した．
- CPU利用率のベース負荷として各クラスタとも30%を設定した．
- クラスタAとBのCPU利用率が逆転する時間帯を設定した．
- クラスタCのCPU利用率は他のクラスタより常に低く設定した．

3クラスタ構成でシミュレーションを行ったのは，2クラスタ構成と4クラスタ以上の構成での振る舞いをカバーできるからである．

シミュレーション結果（図11）からバランス制御の開始から負荷が均等化するまで10インターバル程度かかっているが，その後は均等を維持していることが解る．

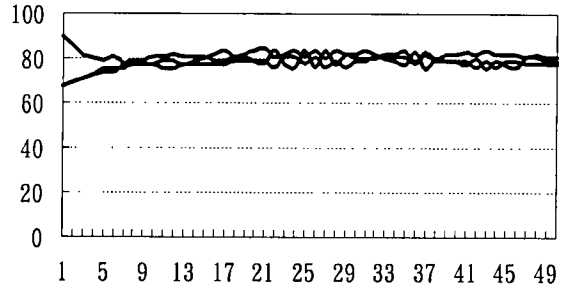


図 11 シミュレーション結果

2.2 バッチの並列処理

バッチ処理のスループットの向上と並列処理について述べる．

CPU バウンド（バッチ処理のうち CPU 処理時間が占める割合が大きいもの）のバッチ処理のスループットは，CPU 単体の性能に依存するため，CMOS 製 CPU 能力が向上して ECL 製 CPU の能力に追いつくまで待たなければならない．しかし，大部分のバッチ処理は I/O 処理時間が占める割合が大きいので I/O 処理時間を短縮することがスループットの向上になる．また，システム全体のスループットを向上するのはバッチ処理の多重度を増やすことも有効である．

バッチ処理のスループットを向上するには，バッチ処理そのものを高速化することが必要となる．シリーズ 2200 ではバッチ処理を高速化するための機能として，EXFILE(メモリ・ファイル：主記憶の一部を大容量記憶域として使う機能)，ESORT(高速ソート)，EXPIPE(ジョブの並列処理：一時的ファイルを使ってデータの受け渡しを行う連続したタスクを並列に実行する機能)が用意されている．この他に XPC のグローバル・キャッシュ機能を使用することによって，さらにバッチ処理時間の短縮を図ることができる．XPC グローバル・キャッシュ機能を適用したシステムではバッチ処理時間を平均 70%短縮した事例がある．

今まで述べたのは，一つのクラスタ内でのバッチ処理を行う場合である．複数のクラスタ間でバッチ処理を並列処理するためには，バッチ処理で使用するデータベースと JCL を共用ディスク上に配置して，どのクラスタからでも参照できるようにする必要がある．また，テープ装置や印書装置などもクラスタ間で共有する必要がある．

並列処理を行う単位は，ラン・グループ（特定の処理を行う一連のラン）単位，ラン単位，プログラム（タスク）単位および，プログラム内の処理単位（バッチレス・システム）に分類できる（表 2)．

この中で，ラン・グループ単位とラン単位での並列処理制御は，現在の運用管理システム (IOF) の変更だけで実現できる．これには，バッチ処理ごとにその負荷を見積もっておき，各クラスタの負荷が平準化するようにバッチ処理を分散してスケジュールする静的な運用管理と，バッチ処理をスケジュールする度に各クラスタの負荷を見て最適なクラスタへスケジュールする動的な運用管理方法がある．当社では静的な運用管理方式を採用した運用管理システム（IOF の新機能）を開発中である．

表2　バッチの並列処理

ラン・グループ単位	異なった処理を行う複数のラングループを別々のクラスタで並列に実行する.
ラン単位	ランの実行順に無関係のランを別々のクラスタで並列に実行する.
プログラム単位	一つのラン内で入出力が独立しているプログラムをランを分割することによって別々のクラスタで並列に実行する.
処理単位	プログラムを最小の処理単位に分割して，トランザクション・プログラム化する.

3. おわりに

実際に並列処理システムに対して負荷平準化機能を適用するには，本稿で紹介した幾つかの方式の中から最適な方式を選択し，または組み合わせて行うことが必要である.

トランザクション量が多いシステムでは，平準化のためのクラスタの負荷を最小にする必要があるため，FEP 方式を採用すべきである．また，端末とクラスタ間のセッションの確立を頻繁に行うアプリケーションでは DCP 方式が採用できる．これらの方式が採用できない場合には，グローバル・キューイング方式やダイナミック・キューイング方式を採用する．クラスタ内で発生するトランザクションに対しては，トランザクション・スケジューラなどのアプリケーション・プログラム自身がクラスタごとの負荷をモニタして，負荷を平準化する制御する方式を採用するべきである.

メインフレームは並列化の流れにある．当社では並列処理を支える負荷平準化機能を持ったプロダクトを継続して提供していく.

参考文献　[1]　日経 BP 社，出揃った汎用並列機とディスク/アレイ　日経ウォッチャー IBM 版 1995.
[2]　阪口喜好, XTC-UDS の概要—密結合から疎結合, そして睦結合へ, ユニシス技報, 第 33 号, 1992, pp. 50〜60.
[3]　「シリーズ 2200 拡張トランザクション機能 XTC 概説書」
「シリーズ 2200 拡張トランザクション機能 XTC 移行解説書」

執筆者紹介　畑　　邦　明 (Kuniaki Hata)
1972 年横浜市立鶴見工業高等学校電子科卒業．同年日本ユニシス(株)入社．以来一貫して OS 1100 の保守および開発業務に従事．この間米国ユニシス社にて 2200/900 および XPC の開発に従事．現在，システムプロダクト部 2200 ソフトウェア開発室に所属.

オープン・プログラミング環境 (OPE) ——OS2200との共存の仕組みと意義

Open Programming Environment (OPE) ——The Mechanism and Significance of Coexistence with OS2200

元山 裕之

要 約 OPEは，シリーズ2200上のUNIX*1OSである．OPEはOS 2200と共存することにより，OS 2200上のユーザの資産(データやプログラム)を継承しつつ，これらの資産とオープンな世界との連携を可能にする．そのため，オープンの世界で標準の各種ネットワーク機能を備えている．また，OS 2200の知識を持たない，UNIXアプリケーション(AP)開発者でも，OPEを使用すれば，シリーズ2200のシステム資源(ハードウェアおよびソフトウェア)を利用するAPを開発できる．

本稿では，OPEの機能概要を紹介するとともに，OS 2200との共存の仕組みや意義を明らかにする．

Abstract OPE is the UNIX operating system running on 2200 Series mainframes. By coexisting with OS 2200, OPE enables inheritance of OS2200 user's properties such as data and programs, and provides the ability to link these properties with the open computing environment. In order to achieve this, OPE provides various network features which are standard in the world of open computing. Use of OPE also allows UNIX applications programmers, who are not familiar with OS2200, to develop applications which utilize both hardware and software system resources of the 2200 Series systems.

This paper gives an overview of OPE features and describes how it is made to coexist with OS2200 and the significance of this coexistence.

1. はじめに

メインフレームを取り巻く環境の変化は激しく，メインフレーマ各社のオープン・システム化が急である．シリーズ2200も例外ではなく，ネットワーク上でUNIX*1やPCと共存することにより，企業情報システムの中核となるオープン・エンタープライズ・サーバとして，生き残りをかけた脱皮を計ろうとしている．本稿で紹介するOPE (Open Programming Environment) は，他のオープン対応プロダクトとともに，オープン・エンタープライズ・サーバの中核を担うキー・ソフトウェアの一つである．

OPEは，シリーズ2200上のUNIXであるが，OS 2200をUNIXに置き換えるものではない．OS 2200の'friend' OSとして，OS 2200と共存していく．OPEが導入された場合にも，既存のOS 2200上のアプリケーション (AP) は，何ら影響を受けずに，そのまま稼働する．機能的に見ても，UNIXと同じオープンなインタフェースでありながら，強力なシリーズ2200のリソース/機能 (ハードウェア，ソフトウェア) を有効利用できる形で実現されており，シリーズ2200上のOSとして，OS 2200と並ぶことのできる機能内容となっている．

本稿では，OPEの機能概要を紹介する (2章) とともに，OS 2200と共存させる仕

組み（3章）や意義（4章）を明らかにしていく．

2. OPE 概 要

OPE は，UNIX System V Release (SVR) 4.0 に準拠している．API (Application Program Interface) は，カーネル，コマンド，シェル，ライブラリとユーティリティ群を含めて，SVR 4.0 と全く一致しているので，UNIX システムのユーザ・インタフェースに慣れている AP 開発者は，何の違和感もなく OPE を使用することができる．

以下に，OPE の概要を AP 開発環境，AP 実行環境，日本語機能，ネットワーク機能および WWW サーバ機能に分けて説明する．

2.1 AP 開発環境

AP 開発者は，Telnet, R-cmd や NFS[*2] を使って，UNIX や PC 上の最新のウィンドウ・システムから OPE を使用できる．Telnet や RLOGIN（リモートなシステムへのログインを可能にする）コマンドを使用することにより，LAN（Local Area Network）ネットワークを経由して，OPE 環境でプログラム開発を行うことができる．NFS を使用すると，OPE ファイル・システムを直接リモート・マウントできるので，PC や UNIX 上のエディタやデバッガ等の AP 開発ツールを使用したプログラム開発が可能となる．また，シリーズ 2200 専用の UTS 端末からのプログラム開発も可能である．図 1 は OPE のプログラム開発環境を表している．

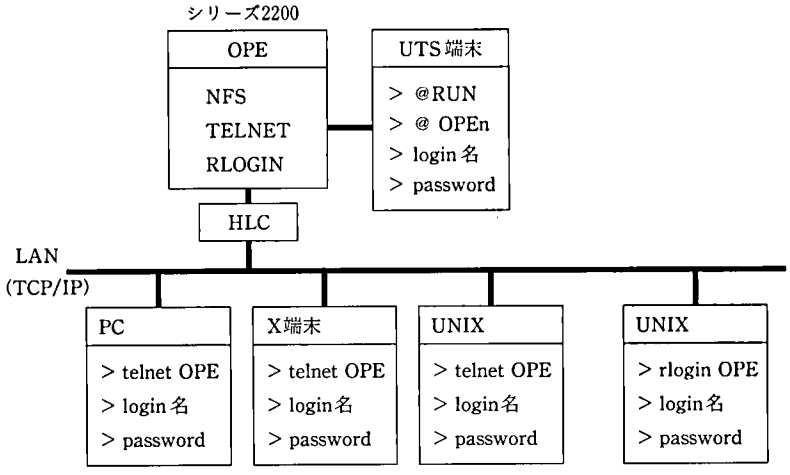


図 1　プログラム開発環境

OPE の AP 開発ツールを使用して，C および COBOL のプログラムを開発できる．ソース・プログラムを開発するには，ed, ex, vi（UNIX のエディタ）を使用する．C プログラムの開発用には，lex, yacc というツールも用意されている．コンパイラは，ANSI C 1989 年準拠の cc コンパイラと，ANSI COBOL 1985 年準拠の cob コンパイラがある．コンパイルされたプログラムは，大規模メモリを使用できる EM (Extended Mode) 環境で実行可能なプログラムとなる．また，SUN ONC[*3] (Open Network Computing) 互換の RPC (Remote Procedure Call) インタフェースのための，C ソース・コード開発用の rpcgen コマンドが提供される．AP 開発システムの仕組みは，

3章を参照のこと.

2.2 AP実行環境

OPEが,UNIX専用のプラットフォームと大きく異なる点は,OPEで開発するAPが,OPE環境で動くUNIXプログラム,OS 2200環境で動くOS 2200プログラム,およびOPE環境とOS 2200環境の両方で動くプログラムの3種類に分けられることである.最終的には,実行形態から次の4種類に分けられる(付記した番号は,図2の番号に対応している).すべて,EM環境で実行される.

① UNIXプログラムでOS 2200連絡なし:OPEのサービスのみを受ける.純粋のUNIXプログラムであり,OS 2200のスケール・メリット(EM環境での大規模メモリ空間,大容量ディスク装置)を前提としたAPとなる.

② UNIXプログラムでOS 2200連絡あり:DMSやRDMS等のOS 2200の付加価値サービスを受けることが可能(Dual).OS 2200上の資産であるデータやプログラムと連携できるAPである.

③ OS 2200プログラムでOPE連絡あり:OPEからPOSIX(IEEEにより制定されたUNIXの国際規格)サービスを受けることが可能(OPEN-API).

④ OS 2200プログラムでOPE連絡なし:従来のOS 2200プログラム.

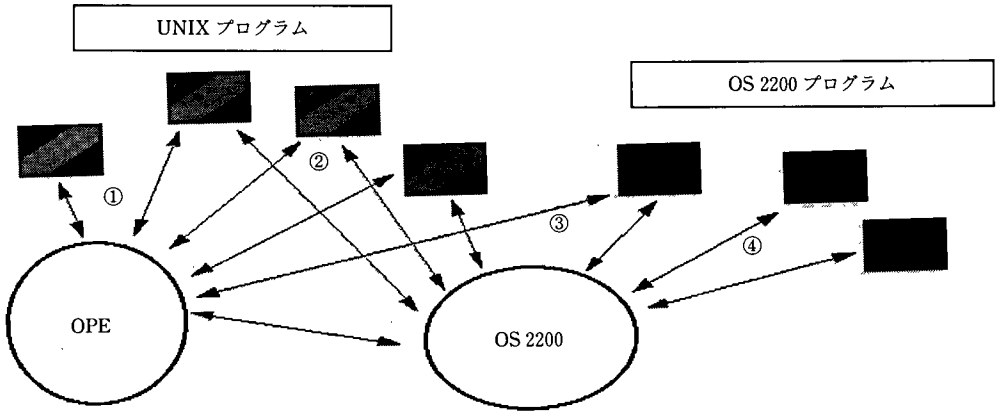


図 2 AP実行形態

AP③および④は,従来,OS 2200の知識がないAP開発者には,開発不可能であった.OPEを使うことによって,UNIXのAP開発者がAP①および②はもちろんのこと,OS 2200のセキュリティ,データ保全性,高速性を最大限に利用できるOS 2200のレガシーAPであるAP④までを開発できるメリットは大きい.たとえば,@制御文を知らないUNIX AP開発者でも,RDMSを使用したHVTIP(High Volume Transaction Interface Package)プログラムの開発ができる.

AP実行形態をこれら4種類のうちどれにするかの決定は,そのAPの使用目的に合わせて,ユーザがプログラム翻訳時に決定できる.たとえば,OS 2200の既存APを段階的にオープン化したい場合には直接的な移行を避け,④→③→②→①の順次に移行が可能である.また逆の方向を進めて,オープンな世界で稼働しているAPを

OS 2200 の中に取り込むことも可能である.

2.3 日本語機能

OPE は, UNIX SVR 4.0 の国際化機能 (Internationalization : I 18 N) をサポートしている. すなわち, 各種環境変数に適切な環境 (ロケール) を設定することにより, そのロケールに合った文字表現や日付表現を選ぶことができる. 具体的な例を見てみると, たとえば環境変数 LANG に japanese を設定し, date コマンドを実行すると,

```
$echo  $LANG
C                                    ←デフォルトのロケールC
$date
Wed Feb 14 14 : 32 : 20 JST 1996
$setenv LANG japanese          ←ロケールを japanese に変更
$date
1996年2月14日(水)14時32分20秒 JST
```

と表示される. date コマンド以外でも国際化されたコマンド, プログラムであればロケールに応じた表示, 動作が行われる.

現在, UNIX システムの日本語文字コードとしては, 日本語 EUC (Extended UNIX Code) コードが主流である. OPE もオープン・エンタープライズ・サーバとして日本語 EUC コードを採用している (将来的に, UNICODE の採用も計画されている). PC や UNIX から見える, OPE の日本語環境は, 日本語 EUC コードを採用している他の UNIX システムと全く同じである (図 3).

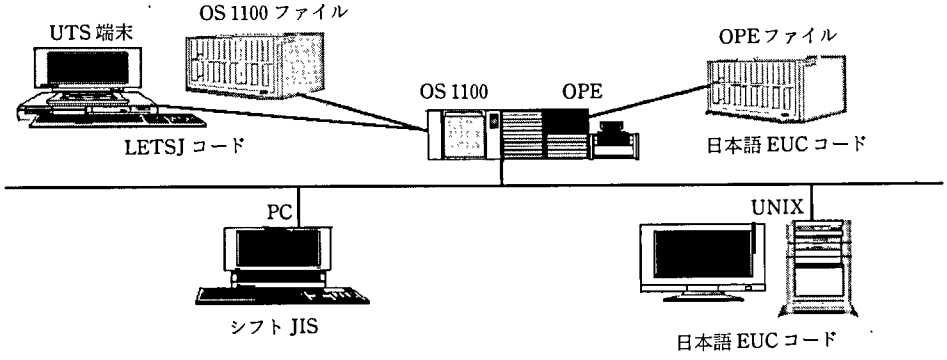


図 3　OPE と日本語文字コード

2.4 ネットワーク機能

OPE は, 次に示すように UNIX を中心とするオープンの世界で, 業界標準となっているネットワーク機能を持っており, シリーズ 2200 をエンタープライズ・サーバとし, UNIX や PC をクライアントとする, クライアント・サーバ・コンピューティングを実現している. これにより, OS 2200 上の資産であるデータや AP とオープンの世界が LAN を経由して, 違和感なく連携できる.

1) RPC インタフェース

SUN ONC 互換の RPC インタフェースをもつ. この機能により, たとえば,

他UNIXシステム上のプログラムから，OPE上のHVTIPプログラムを呼び出すことが可能となる．また，RPC機能を持つクライアントとサーバのプログラム間で様々な情報の交換を行うこともできる(異機種間のプログラム通信)．クライアントからリモートマシン上に存在するプロシージャをコールすることが可能となる．

2) NFSによるファイル共有

NFSのサーバ機能により，シリーズ2200の大容量なディスク領域上のファイルを他のマシンから共有し，利用することができる．シリーズ2200上の財産である各種データを，オープンの世界で流通しているソフトウェアで直接処理可能となる．また，NFSクライアント機能により，他のマシンのディスク領域をOPEからファイル共有し利用することもできる．

3) Telnet機能

TCP/IP*4接続されたターミナル(PCやUNIXワークステーションを含む)からtelnetによる接続をサポートする．

4) バークレー版のリモート・コマンド

他のUNIXシステムからのrlogin，rcp，rcmd，rsh，rshdというリモート・コマンドの要求を受け付けることができる．

5) FTPによるファイル転送

TCP/IPに則した業界標準のFTP機能を使用できる．uドライバ機能により，シリーズ2200上のSDFファイルおよびプログラム・ファイルを直接ファイル転送の対象にできる．将来的には，当社の高速ファイル転送機能であるCPFTP (Cooperative Processing FTP) インタフェースも提供する予定である．

6) APレベルインタフェース

ソケット，TLI (Transport Level Interface) の二つのインタフェースをサポートする．

2.5 WWW (World Wide Web) サーバ機能

OPEの大きな機能としてWWWサーバ機能がある．標準的なWebサーバとproxyサーバの機能を持つCERN Webサーバである．この機能を使用することにより，シリーズ2200のOPE上にホームページを持ち，WWWスタイルの情報をインターネットに発信することが可能となる．シリーズ2200のスケールメリットを活用した大規模なWebサーバとなる．

クライアントであるUNIXやPCからは，NetscapeやMosaic等のWWWブラウザを使用してOPE上のホームページを見ることができる．クライアントからの要求に応じて，該当するホームページのHTMLファイルやイメージファイルをクライアントに送信する．受け取ったファイルを解釈してクライアント画面に表示するのは，ブラウザの役割である．

HTMLだけのホームページ (Static HTML) は静的な情報の発信となり，HTMLに書かれた内容以上の情報は発信できない．しかし，CGI (Common Gateway Interface) と呼ばれるインタフェースを用いれば，サーバ上で動く各種APとの連携を行うことが可能になり，APで動的に生成したHTMLファイルを，ブラウザに送ることが

できるようになる (図 4). CGI は，ブラウザでの対話的な処理を行う場合 (ユーザが入力したキーでデータベースを検索し，結果をブラウザに表示する等) に有効である. OPE Web は, CGI を提供し, OS 2200 の RDMS 検索等の AP 処理と連携した Web サーバとなることができる.

今後，CGI を用いた RDMS AP や TIP AP はもちろんのこと，java 言語 AP との連携が増加すると考えられる. java AP は，サーバ上に存在し，Hotjava 等の java 対応の WWW ブラウザ上にダウンロード後，実行されるので，マシンや OS に依存しない. OPE Web 機能は，OS 2200 上の資産との連携という OPE の特徴を代表する有効な機能である.

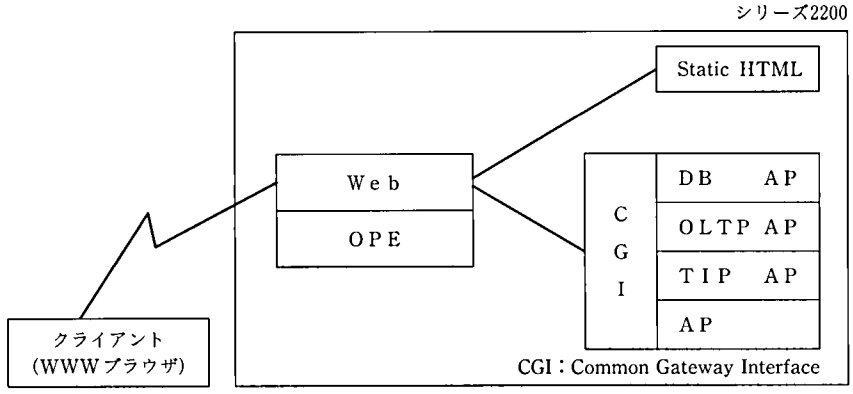


図 4　OPE Web と AP の連携

3. OPE と OS2200 共存の仕組み

OPE は，OS 2200 上のアプリケーションの一つであり，EM (Extended Mode) サブシステムとして構築されている. OS 2200 上に最大 36 個の OPE を導入できる. OPE は，OS 2200 と共存することによって，OS 2200 の機能を最大限利用できるよう工夫されている. OPE がどのような仕組みで OS 2200 と共存するのかを示す.

3.1 OPE の扱うファイル

OPE で扱うことができるファイルは大きく 2 種類に分けられ，ともにシリーズ 2200 のディスク装置上のファイルである. 一つは，UNIX のファイル・システムと同じ階層的なファイル構造を持つ OPE ファイルシステムである. もう一つは，OS 2200 の SDF (System Data File) ファイルおよびプログラム・ファイル中のシンボリック・エレメントであり，OPE の u ドライバ機能 (OPE カーネルの持つデバイス・ドライバの一つ) で直接参照できる. 図 5 に，OPE で扱える 2 種類のファイルを示す.

それぞれのファイルについて，詳しく解説する.

一つの OPE ファイル・システムは，一つの OS 2200 ファイル中に構築される. たとえば，OPE ファイル・システムを作成し，その中に mydir というディレクトリ，さらにその下の階層に test というディレクトリを作成する場合，次のようにする.

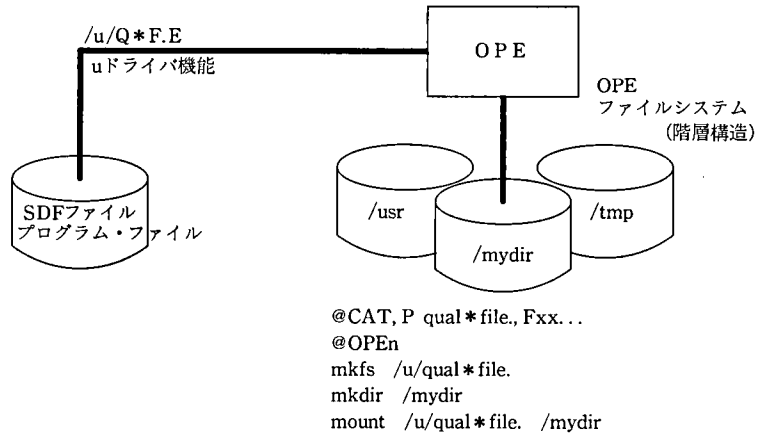


図 5 OPEの扱うファイル

```
>@CAT, P  qual * file., Fxx.....   ……OS 2200 ファイルのカタログ
>@OPEn                            ……OPE 呼び出し
>login：>＊＊＊＊                 ……ログイン識別名入力
>passwd：>＊＊＊＊＊＊＊          ……パスワード入力
>mkfs  /u/qual * file.            ……OPE ファイル・システム作成
>mkdir  /mydir                    ……デレクトリの作成
>mount  /u/qual * file.   /mydir  ……両者をマウント
>mkdir  /mydir/test               ……階層デレクトリの作成
```

これらのファイルはOS 2200 ファイルの中に存在するため，信頼性のあるOS 2200のファイルセーブおよびリカバリ機能（FAS 1100）によって保護することができる．

uドライバ機能で，プログラム・ファイルQ＊F中の，シンボリック・エレメントEを参照する例を次に示す．

```
>@OPEn                            ……OPE 呼び出し
>login：>＊＊＊＊                 ……ログイン識別名入力
>passwd：>＊＊＊＊＊＊＊          ……パスワード入力
>cp  /u/Q * F. E  /mydir/pf. txt  ……OPE ファイルにコピー
>vi  /u/Q * F. E                  ……vi エディタで直接編集
>cat a. txt b. txt>  /u/Q * F. E  ……リダイレクトの出力先
```

(注)上記二つの例は，UTS 端末からの使用例である．PC や UNIX から使用する場合には，ログイン識別名の入力からとなる．

3.2 AP 開発システム

OPEのコンパイラは，OPE独自のコンパイラではなく，OS 2200上で実績があり，安定性の高いUCS (Unisys Compiling System) コンパイラを内部的に使用している．ccコンパイラは，UCS Cを，cobコンパイラは，UCS COBOLをそれぞれ内部的に使用している．また，連結編集用のldユーティリティは，リンキング・システムを使用している．これにより，EM環境で実行可能なプログラムを開発できる．

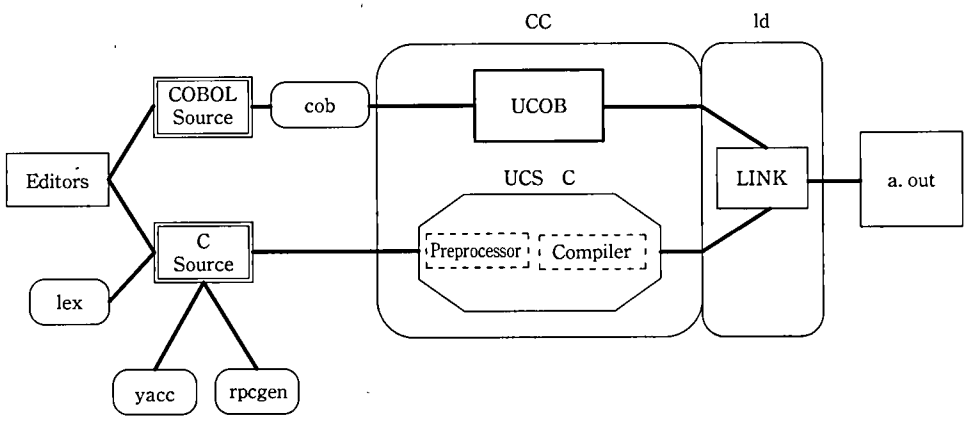


図 6　プログラム開発システム

図 6 は，OPE のプログラム開発システム（編集・翻訳・連結編集）を表している．

3.3　AP 実行システム

OPE で開発した 4 種類の AP を実行する仕組みを，図 7 に示す．図中に付記した番号は，(2.2 節 AP 実行環境) で述べた番号に対応している．

OPE は OS 2200 とともにカーネルとして作動する．UNIX AP からは，OPE は標準 UNIX シェル環境に見える．ハードウェア・リソース割り当て，スケジューリング，ディスパッチング，メモリ管理，フォルト・ハンドリングといった中核の機能は，OS 2200 カーネル部分で処理される．高速コミュニケーションは CMS/TSAM インタフェースを通して行われる．このような構造のため，シリーズ 2200 のすべてのシステム資源(ハードウェア，ソフトウェア)が OPE から利用可能となっている．その結果，シリーズ 2200 の性能 (機能/効率) が上がるとともに，OPE の性能も向上する仕組みになっている．

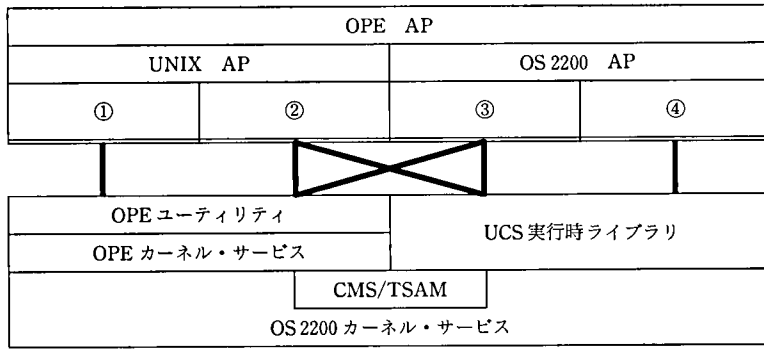


図 7　実行時システム構造

3.4　日本語自動コード変換システム

OPE は，日本語文字コードとして，日本語 EUC コードを採用している (2.3 参照)．しかしながら，OS 2200 では日本語文字コードとして従来から独自の LETS-J コード

を採用し，OS 2200 専用のプリンタや端末などの周辺機器はすべて LETS-J を前提としている．したがって OPE は，OS 2200 と共存するために，次のような LETS-J 対応を行っている．

1) UTS 端末使用時の自動コード変換

UTS 端末からの LETS-J による日本語入出力を，日本語 EUC コードあるいはシフト JIS コード等，そのときのロケールに合った文字コードへ，あるいは文字コードから自動的に変換する．この機能により，UTS 端末使用時でもユーザは LETS-J 変換を行わずに日本語文字を表示あるいは入力できる．

2) OS 2200 プリンタ使用時の自動コード変換

次の 2 通りの方法で OS 2200 プリンタへファイルを出力することができる．

・プリンタ・スプール（LP）機能を用いて OS 2200 プリンタに印書する．

・lpr コマンドで OS 2200 プリンタ・スプールに直接印書する．

いずれの場合も OPE ファイルは，一度 OS 2200 ファイルのシンビオント・ファイルにコピーされるが，LETS-J コードへの変換は各々のコマンドが自動的に行うため，ユーザは LETS-J コードへの変換を行う必要はない．

3) demand/submit コマンド使用時の自動コード変換

demand，submit コマンドは，OPE セッションから OS 2200 のデマンド・ランを起動するコマンドである．これらのコマンドも内部的に LETS-J コードへの変換を自動的に行うので，ユーザは変換を行う必要はない．

4) u ドライバの自動コード変換

LETS-J コード前提の OS 2200 ファイルを扱う場合，u ドライバが LETS-J コードへの自動変換を行う．ユーザは，コードの違いを意識せずに OS 2200 ファイルを扱える．

5) コンパイラの EUC コード処理

OPE が内部的に使用する UCS コンパイラの本体は，もともと日本語コードに依存しないようデザインされている．cc や cob では，EUC コードを処理できるよう OPE 側でソース・プログラムの読み込みやコンパイル・リストの出力を行っている．

6) コード変換ツールの提供

次の 4 種類の日本語文字コード相互の変換についてコード変換ツールを提供する．

・LETS-J コード
・日本語 EUC コード
・シフト JIS コード
・JIS コード

3.5 ネットワーク環境

OPE カーネルのネットワーク・モジュール構造は図 8 のとおりである．OS 2200 の CMS 2200 を経由して，オープンな世界と結合している．

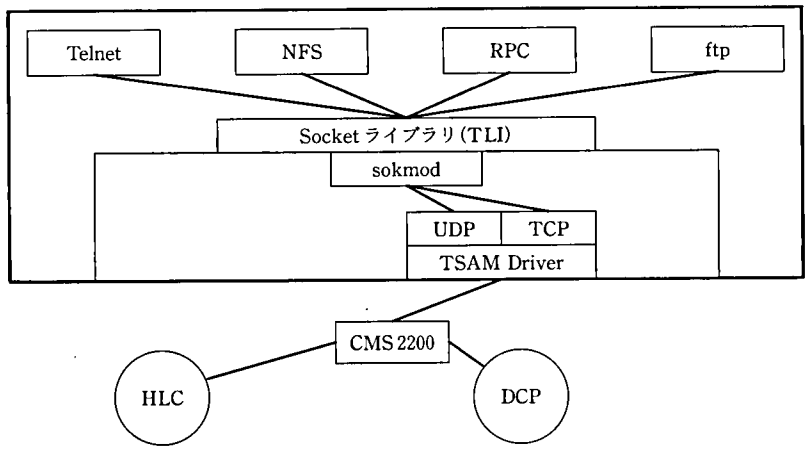


図 8　OPE カーネルのネットワーク・モジュール

4. 異種 OS 共存の意義

オープン・システムが満たすべき条件である相互運用性，移植性，および拡張性の各観点から，OPE の有効性を述べることで，異種 OS 共存の意義を明らかにする．

4.1 相互運用性

現在，UNIX システムをサーバとした，クライアント・サーバ・システムは，サーバを OPE に切り替えるだけで再構築できる．また，OPE は OS 2200 と共存し，OS 2200 上のデータや AP を容易に参照できる．これにより，エンドユーザはクライアントから，従来通りのオープンな共通のインタフェースで，OS 2200 上のデータや AP と容易に連携できるようになる．相互運用性を向上させる，いくつかの例を示す．

SUN ONC 互換の RPC インタフェースのため rpcgen ユーティリティを使用すると，クライアントとサーバのどちらの RPC インタフェース部分も簡単にできる．この機能により，たとえば，他 UNIX システム上のプログラムから，OPE を通して，OS 2200 上の HVTIP プログラムを呼び出すことが可能となる．また，RPC のクライアントとサーバのプログラム間で様々な情報の交換を行うこともできる．

また，NFS 機能を使うことにより，NFS を実装している UNIX や PC から，OPE ファイル・システムをリモート・マウントすることができる．OS 2200 上のファイルは，OPE の CP コマンドで OPE ファイル・システムに自動変換されるので，UNIX や PC 上のソフトウェアを使って OS 2200 上のデータを直接処理することができる．また逆に，OPE をクライアントとして，UNIX や PC のファイル・システムをマウントし，利用することもできる．

4.2 移　植　性

OPE は，プログラムの移植を容易に行うことのできるプログラム開発システムを提供している (図 6)．SVR 4.0 および ANSI 標準規格 (C 89，COBOL 85) に準拠したプログラムから成り，UNIX のデータベースに依存しない AP であれば，OPE 上でコンパイルし移植することができる．移植された AP は，OS 2200 が持つスケーラビリティ，セキュリティ，データ保全性等のエンタープライズ・レベルの機能を享受できる．

たとえば，フリーソフトの sendmail（電子メイルの送受信システム）を OPE 上に移植することにより，既存の UTS 端末から，インターネット経由で，他システム間と電子メールのやりとりを行うことが可能となる．また，compress コマンド(ファイルの圧縮･復元プログラム）を OPE 上に移植することにより，OS 2200 上のデータを対象に，ファイルを圧縮し，UNIX や PC にファイル転送後，復元するといった，UNIX で通常行われている使い方ができるようになる．

またこの逆に，シリーズ 2200 の豊富な資源を活かしながら，OPE 上で開発した AP を，他の UNIX システム上に移植することもできる．OS 2200 上の AP を他システムへ直接移植することは，一般的に困難であるが，OPE 上の AP を他システムへ移植することは容易である．

4.3 拡　張　性

OPE では，他の UNIX システムにはないシリーズ 2200 のスケール･メリットを利用し，システムを拡張することが可能である．

OPE ファイル･システムはシリーズ 2200 の大容量なハード･ディスク上に作成される．そのため，新たな OPE ファイル･システムの追加は，ディスクの増設をすることなく，OS 2200 のファイルを割り当てること（@CAT）により可能となる．

一つのシリーズ 2200 上に，最大 36 の異なるモードの OPE システムを導入することが可能である．これは，UNIX システムを 36 台稼働させることと同等である．テスト用，本番用等利用目的別に分けたり，部署ごとに OPE システムを運用するということもできる．また，一つの OPE システムを，ある AP 専用で使用することも可能となる．たとえば，モード A の OPE システムは，OPE の Web 専用に使用するということもできる．複数の OPE システム間は，UNIX のネットワーク機能で接続されている．

もちろん，シリーズ 2200 のすべてのシステム資源（ハードウェア，ソフトウェア）が，OPE から利用可能である．したがって，シリーズ 2200 のハードウェアやソフトウェアの機能向上や新機能追加を，OPE 利用者も享受することができる．

5. お　わ　り　に

現在，メインフレーマ各社は，「オープン･システム化の進展中，メインフレームのあり方」をユーザから問われている．OPE は，これに対する当社の回答の一つである．すなわち，ユーザが OS 2200 上に蓄積している資産であるデータやプログラムを活かし，継承しながら，オープンな世界からこの資産と連携できるようにするプロダクトとして OPE を，OS 2200 と共存できる形で提供する．

今後は，他のオープン対応プロダクトと共に働き，シリーズ 2200 がクライアント･サーバ･システムのオープン･エンタープライズ･サーバとして確固たる位置を占めることができよう，最新の標準に準拠した機能向上を適宜実施していく必要がある．また，ユーザにとって有効なソフトウェアを OPE 上に移植あるいは開発し，利用してもらえるようにすることが我々の使命であると考える．

*1 UNIX は，X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
*2 NFS は，Sun Microsystems 社の登録商標である．
*3 ONC は，Sun Microsystems 社の登録商標である．
*4 TCP/IP は，テキサス・インスツルメント社の登録商標である．
*5 WindowsNT は，Microsoft 社の商標である．
その他，本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

参考文献

[1] Wayne J. LeBlanc, "Enterprise Web Servers—Expanding Business Opportunities", UNITE, Fall 1995.
[2] Gerald Hupperts, "The Amicus Overview—An Open 2200 Enterprise Server Environment", UNITE, Fall 1995.
[3] Joellen Sleeter, "AMICUS Commands and Utilities—What Are They ?", UNITE, Spring 1995.

執筆者紹介

元 山 裕 之 (Hiroyuki Motoyama)
1978 年横浜国立大学工学部生産工学科卒業．同年日本ユニシス(株)入社．主として，UNISYS 2200 シリーズの言語プロセッサの開発，保守業務に従事．現在，システムプロダクト部 OES 開発推進室に所属．SC 22/COBOL WG 委員．SC 22/POSIX WG 委員．

Aシリーズの次世代オープン・エンタープライズ・サーバの実装技術

Technology for Implementing Next-Generation Open Enterprise Servers for A Series

室木　通, 原　広仁

要　約　AシリーズではこれまでTCP/IP関連製品，Open OLTP，ODBC*1など，いわゆるオープン製品やCCE/CCPと呼ばれる独自の協調コンピューティング環境を提供することで，WindowsおよびUNIX*2との協調処理環境に積極的に対応してきた．

しかし市場でのこのような製品への要求も徐々に高度化してきている．とくに昨今のMicrosoft製品の影響は大きく，メインフレームのオープン性という面でも必然的にWindows/Windows NT*1との，より高い接続性，協調処理が重要になってきた．

次世代オープン・エンタープライズ・サーバはこのような市場を背景として生まれた．この製品はまだ成長段階であるが，これまでのメインフレームのオープン性という言葉のイメージを一歩踏み越えたWindows/WindowsNTとの接続性，協調処理を実現しようとしている．本稿では，このオープン・エンタープライズ・サーバの目指すところを述べ，ハードウェア/ソフトウェアでの実装技術，さらに将来のメインフレーム像について触れていく．

Abstract　The A Series has shown active participation in cooperative computing environments with UNIX and Windows by providing so-called open products such as TCP/IP-related products, OpenOLTP and ODBC, and by providing a proprietary cooperative computing environment named CCE/CCP.

However, market demand for these types of products is becoming gradually sophisticated. Especially, being greatly influenced by recent Microsoft products, high connectivity and cooperative computing with Windows and WindowsNT inevitably became the deciding factor in the context of mainframe "openness".

The next-generation open enterprise server was developed with this market requirement. Although there still is room for more development, this product strives to go one step ahead of what we imagine as an open mainframe by providing connectivity to and a cooperative computing environment with Windows and WindowsNT. This paper discusses the goals of this next-generation open enterprise server with actual hardware and software technology, and briefly mentions future expectations for mainframe technology.

1. はじめに

現在のメインフレームにおいては，新しく導入される端末にPCが採用されることが一般的になってきた．情報化白書95によれば，全産業計で56.9%が専用端末からPCへの移行を実施している[1]．最近のC/SS（クライアントサーバシステム）の文献を見ても，メインフレームは「ホストシステム」という呼称によって，この対象外に位置付けられがちであるが，基幹業務データが存在するメインフレームを，UNIX*2やWindows NT*1機のような「サーバマシン」として使用したいという要望は存在している．これはインターネットにおいてでさえ例外ではない[2]．

一方，最近の高処理能力用途のPCサーバは，PC供給元が言うところの「メインフレーム機能」を取り込むべく，250から600 MIPS前後のプロセッサ，対称型マルチプロセッサ（Symmetric Multi Processing），ECCメモリの採用，冗長なシステム構成，非同期入出力化など，自身の構成要素（プロセッサ，メモリ，入出力）の「堅牢性（Robustness）」の強化を図りつつあると見てよい[3]．

1990年代の初頭から当社は，基幹業務に必要な堅牢性を備えたサーバとして「エンタープライズ・サーバ」の方向性を打ち出してきたが，ここで紹介する「次世代オープン・エンタープライズ・サーバ（以下OESと略す）」においては，特にWindows NTオペレーティングシステム環境との親和性が従来製品に比べ格段に向上する．OESはPC供給元とは逆の，すなわち，メインフレームがPCを取り込むというアプローチを採用した．

本稿では，Aシリーズのオープン対応の事例として，このOESの特徴や実装について紹介すると同時にその意義を考察する．

2. 異なるアーキテクチャの融合

2.1 OES出現の背景

Aシリーズが示していたオープンの定義は，国際標準（POSIXなどのX/Open規格，OSI規格等）への準拠と業界標準（TCP/IP, NetWare*3, SNMP, PostScript*4等）への準拠が柱であったが[4]，PCの台頭に伴い，UNIXの時代には議論されなかった新しい業界標準「マイクロソフト製品」が出現した．現実に目を向けると，現在のアプリケーション構築は，GUI（グラフィカルユーザインタフェース）とRAD（Rapid Application Development）ツールの出現により，Windows上でのC/SSを抜きには考えられなくなってきている．また，インターネットアプリケーションも含め，多くのソフトウェア資産がWindows上で利用可能になってきていることから，当社はUNIXに次いでマイクロソフト製品を「積極的に」オープンであると認める方向に転換した．このように業界標準が国際標準より優先する事例は，OSIとTCP/IPとの関係にも見られる．

2.2 OESの位置づけ

OESは，AシリーズとWindowsNTを結合させるためのソフトウェア/ハードウェア製品である．これはメインフレームであるAシリーズとWindowsNTが並列に実行される形態である．コンピュータの並列結合の分類には，以下のような分類法がある．

1) コードストリームとデータストリームの多重度によるフリン（Flynn）の分類
2) リクエスタ（プロセッサ）対メモリの構成によるショア（Shore）の分類
3) プロセッサ間の結合の形式・度合による分類
4) 分割されるタスクの大きさ（タスク粒度）による分類[5]

これらの分類は，本来，同質のアーキテクチャ（ないしプロセッサ）同士についての言葉であるが，ここでは，3)と4)の分類に基づいてOESの位置づけを考えてみる．

2.2.1 プロセッサ間の結合の形式・度合による分類

この分類によれば，OESは「疎結合マルチプロセッサ方式」となるが，メモリが別

であることから，分散メモリ型並列（DMPP）とか単にクラスタリングと呼ばれる場合もある．しかし，ここでは分散結合される対象が異種プロセッサであることに注目し，HMP(Heterogeneous Multi Processing)と呼ぶことにした．HMP は，UP(Uni/Single Processing)，SMP (Symmetric Multi Processing)，MPP (Massively Parallel Processing) に次ぐ新しい結合方式である．このような四つの結合方式を統合し一つのシステムとして機能させる技術を，ここでは「クラスタリング」と呼ぶ．クラスタリングは，一般的に以下のメリットをもつ．

- スケーラビリティ
- 連続運転
- 柔軟な構成
- 自動的な障害検出
- シングルシステムイメージ
- ワークロードバランスの最適化
- 災害時被害の最小化

クラスタリングは決して新しい手法ではなく，古くは Digital 社が VAX 機同士をクラスタ構成（同種のクラスタ結合）した例がある．

HMP はメモリや補助記憶装置を独自に持ち，メッセージの送受によって並列的な論理結合が行われる．並列ということでは，データベースの並列アクセス技術に代表される MPP 形態での無共有 (Shared-Nothing) 方式があるが，HMP は異種プロセッサの結合であるという以外に論理的に資源を共有する点が異なる．HMP を実現した OES は，図 1 にあるように，A シリーズと Windows NT 機をトランスピュータ型の高速バスで結合したもので，クラスタリングにおける柔軟な構造，シングルシステムイメージ，ワークロードバランスの最適化の利点に加え，

- コストメリット
- オープン性
- 2 層ないし 3 層の C/SS 向け構成

などの利点を得ることができる．

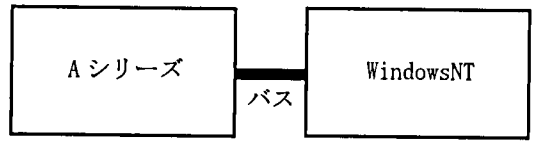


図 1　OES における A シリーズと WindowsNT の結合(1)

2.2.2　タスク粒度による分類

もう一つ，タスク粒度による分類から考えてみる．タスク粒度とは純粋にソフトウェア面からの分類であり，分割される処理単位の大きさによって区別を行うものである．一般に，分散メモリ型結合ないしクラスタリングにおいては，細粒度におけるメッセージハンドリングの低レイテンシ（潜在的遅延）を，オーバーヘッドの削減とメッセージハンドリング実装技術により，いかにして得るかということが課題となって

いる[6]．

この課題については，今回のOESのように，プロセッサアーキテクチャが一意であるAシリーズと，複数のプロセッサアーキテクチャ上で実装されるWindows NT機の間の結合では，演算レベルの細かい粒度の結合は意味を持たないと考えられることと，一般的なC/SSにおける処理が，SQL文の処理といった粗い実行単位が中心であると考えられることから，大きな問題ではないと判断している．C/SSモデルによるシステム構築が黎明期であった時期は，検索された全てのレコード集合をクライアント側に取り込むような中程度の粒度の結合も実践されたが，それがLANのトラフィックを圧迫し，結果的には「ストアドプロシジャ」「トリガファンクション」と呼ばれる集中的なマクロ手続き方式を生むことになった．このことから，OESが実現しているタスク粒度は，今日のC/SSを扱うのに適切なものといえる．

2.2.3 結合の手段SMB

一般的に，異なったアーキテクチャについてはTCP/IPソケットを介した結合に見られるように，トランスポートレベル層の結合であることが多い．ところがOESについては，さらに一歩踏み込んで，サーバメッセージブロック（SMB）プロトコルを用いてセッション層以上で結合する形態を取ることができる．

SMBプロトコルは，PCインターネットワーキングにおける利用者レベルと共有レベルに関するX/Open標準プロトコルであり，C/SSの核となる機能を提供する[7]．SMBプロトコルを用いてマイクロソフトネットワーク（ないしWindows NTドメイン）に結合することは，資源を共有しつつルーズな結合を行う，すなわちプロセッサの「異種クラスタ結合」を可能にすると考えてよい（図2）．

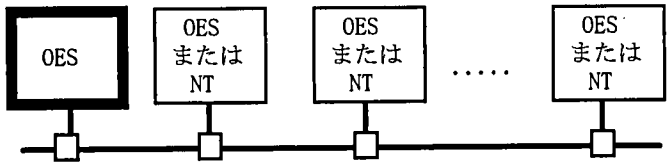


図2　OESにおけるAシリーズとWindowsNTの結合(2)

前述の，粒度の細かさが引き起こす問題の解決に関する実装，例えばメッセージ送受におけるコントロール部分との制御のオーバラッピングを許すかどうか等はX/OpenのSMB自体は規定していない．言い換えれば，この分野は供給側の実装に任されているといえる．バス型の結合を想定しているOESは，必要に応じてさらに細かい粒度に発展させる事も可能である．

2.2.4 サービス

この結合方式の元で実現される上位層のサービスは「WinServices」と呼ばれ，まず

- ファイルサービス
- プリントサービス
- 名前付きパイプ，WinSockの完全サポート

があり，後続のリリースでは，

- ユニバーサルメッセージ交換（MS-Exchange）

・OLE（Object Linking Embedded）

・マイクロソフト RPC（Remote Procedure Call）

等が計画されている．

結果的に，HMP 結合によって A シリーズから見ると，WindowsNT が A シリーズの手足に見え，WindowsNT から見ると，A シリーズは WindowsNT のネットワーク資源に見える．例えば，A シリーズファイルを Windows NT のファイルマネジャや Windows 95 のエクスプローラを用いて移動や削除などの操作ができるようになったり，従来，MAKEUSER と呼ばれていたユーティリティで行っていた A シリーズのユーザ管理を，「ユーザマネジャ」と呼ばれる Windows アプリケーションで行うことができるようになる．

次章では，この実装について詳しく紹介する．

3. OES の実装

3.1 ハードウェア構成

図 3 に OES のハードウェア構成を示す．点線内の WindowsNT サーバと A シリーズオペレーティングシステム MCP（以下 MCP と呼ぶ）が動作するプロセッサ，および一組のモニターとキーボードが最も基本の構成である．

ハードウェア面から HMP を支える最も重要な技術は二つのプラットホームをつなぐ CS-Bus と PCCA カードである．CS-Bus とは，A シリーズプラットホームで使用される入出力チャネルバスのことで A シリーズ標準のバス仕様である．また PCCA カードとは WindowsNT 側プラットホームの業界標準のバスに挿入され，A シリーズのバスと WindowsNT プラットホームのバスを整合するハードウェアである．

この CS-Bus と PCCA カードを使用して二つのプラットホームを接続することで，プラットホーム間の高速でオーバヘッドの少ないデータ転送を可能にしている．

このバス接続では利点を生かした多種の応用が可能であるが，一例として CS-Bus

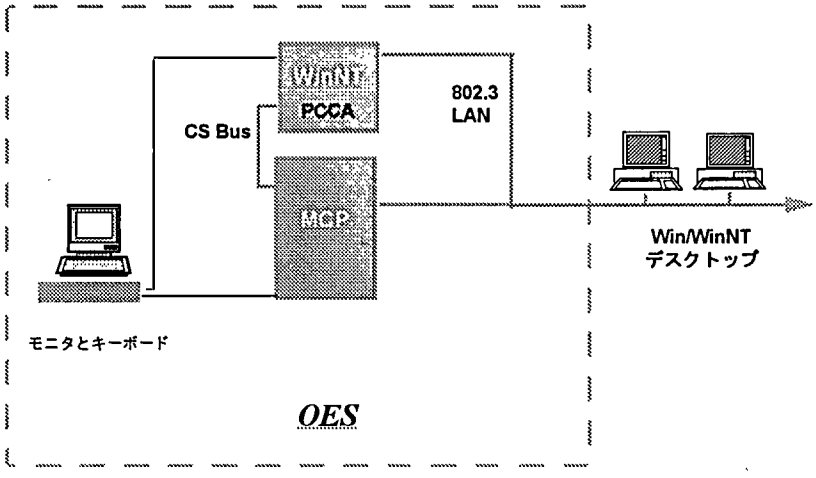


図 3　OES のハードウェア構成

経由のTCP/IP接続（仮想TCP/IP）について述べる．この仮想TCP/IP環境では，TCP/IP上のアプリケーションにはLAN接続のTCP/IP環境と全く同じ動作を保証している．一般的なLAN環境と比べた利点として，①下位のプロトコルが省略されたことで双方のプロセッサ負荷を軽減，②バスを使うことで，高いデータ転送能力が期待できる，③双方のプラットホーム間が直接かつ単独に接続されることで，高いデータの信頼性とセキュリティが期待できる．

3.2 マイクロソフトネットワーク

HMPの重要な要素の二つ目は，異なるオペレーティングシステム環境をシームレスに接続することである．OESではマイクロソフト標準を取り入れることで，MCPとWindows/WindowsNTデスクトップのシームレスな統合を実現している．

WindowsやWindowsNTのネットワーク製品には，サーバメッセージブロック(SMB)プロトコル[8][9]が使用されている．OESはMCPとWindows/WindowsNT環境のシームレスな接続を行うため，MCP環境にも標準機能としてTCP/IP上にSMBプロトコルの支援を実装している．この結果，デスクトップ環境から見るとMCP環境は一つのマイクロソフト互換サーバとして存在することになる．例えばデスクトップ環境からネットワークをブラウズすると，MCP環境は一つのサーバとして現れ，そのMCP制御下の資源はWindowsNTサーバにある資源と同じ方法で利用することができる．

ところでSMBは，Windows NT, LAN Manager, Windows for Workgroupsなどで各ノード相互間で通信を行うとき使われ，X/Openで定義されたプロトコルである．このプロトコルは約100種類定義されていて，以下の四つのインプリメントレベルに分けられている．

core protocol
core plus protocol
extended 1.0 protocol
extended 2.0 protocol

これらのプロトコルは，後のプロトコルがその前のプロトコルを包含して，core plus protocolはMICROSOFT NETWORKS 1.03, extended 1.0 protocolはLAN-MAN 1.0などに対応している．core protocolは最も基本となるプロトコルで，これはさらにカテゴリ別に以下の四つに分類されている．

Connection Management Requests
File Operation Requests
Directory and Attribute Operations
Spool Operation Requests

例えばConnection Management Requestsには通信するプロトコルレベルを決める“SMBnegprot”が含まれている．また，File Operation Requestsに含まれる“SMBopen”は，すでに存在するファイルのオープンを行うプロトコルである．

core protocol以降のプロトコルでの拡張については，例えばcore plus protocolではFile Operations Requestsが拡張され，またextended 2.0では拡張属性(Extended Attribute)やロングファイル名（long filename）が支援されている．なおOESはプ

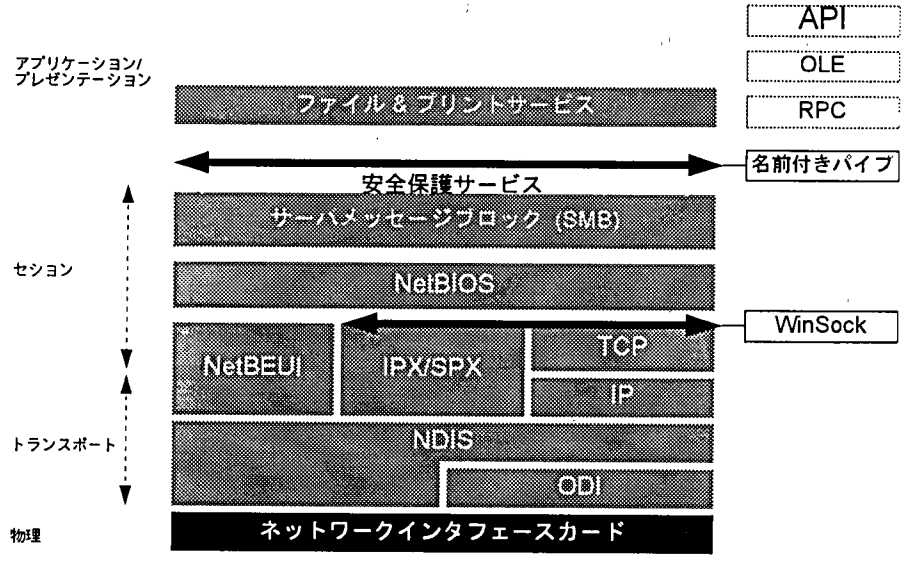


図 4　マイクロソフトネットワーキング

ロトコルレベルとして SMB extended 2.0 protocol を支援している.

OES では,このような SMB を実装することで Windows/WindowsNT デスクトップとシームレスにファイルサービス,プリントサービス,名前付パイプなどを実現している.図 4 にマイクロソフトネットワーキングのトランスポート層,セッション層,アプリケーション/プレゼンテーション層を示す.OES の初期製品では,名前付パイプ,WinSock を実装し,さらに後続の製品で,マイクロソフト RPC,OLE を実装する予定である.またマイクロソフトネットワーキングの採用は,MCP 環境への PC の接続性もあわせて改善した.PC 利用者はかつてのように MCP 環境とのネットワーク接続に困難さを感じるようなこと,たとえば A シリーズ固有の知識や操作が必要になったり,A シリーズ上でのデータ通信の構成を強いられたりすることがなくなった.

3.3　ファイルサービス/プリントサービス

3.3.1 ファイルサービス

ファイルサービスとは,複数のシステムから一つのファイルを共有できるようにすることである.たとえば Windows/WindowsNT 環境ではファイルマネージャを使ってローカルディスクをアクセスするのと同じ方法で,ネットワーク上のファイルサーバのディスクをブラウズしたり,そこに有るファイルをコピーしたりすることができる.

OES ではこれと同じファイルサービスを MCP 制御下のディスクで実現している.たとえば,OES とネットワーク接続された Windows/WindowsNT デスクトップ環境からファイルマネージャを起動し,ネットワーク接続を使って MCP 制御下のディスクをネットワークドライブ接続することができる.もしそのディスク上に,MCP 制御下のアプリケーションやユーティリティなどで作成したファイルがあれば,そのフ

ファイルを見ることができる．同様にWindowsのメモ帳（NotePad）を起動し，MCP制御下のアプリケーションで作成したテキストファイルをオープンして内容を編集することができる．逆に，Windows/WindowsNTデスクトップ上のソフトウェアでこのディスク上にファイルを作成し，MCP制御下のアプリケーションから使用することも当然可能である．

OESではMCP制御下のファイルシステムをそのままマイクロソフトネットワーク環境に提供することで，双方のオペレーティングシステム環境のアプリケーションからディスクファイルを直接アクセスすることができるようにしている．この方法はWindowsのファイルシステムをMCP制御下のディスクに実現する，いわゆる仮想ディスク方式ではない．

3.3.2 共有ディレクトリへのアクセス

OESでは，ファイルサービスにおける共有ディレクトリへのアクセス制御として二つの方式を用意している．一つは，MCP環境でのファイルアクセス規則に従った方法で，ノーマル共有アクセスと呼ぶ．もう一つは，Windows/Windows NT環境のファイルアクセス規則に従った方法で，パブリック共有アクセスと呼ぶ．ただしこのパブリック共有アクセスもMCP制御下のファイルシステムを共有するので，MCPのファイルアクセス規則と無縁というわけではない．なお，これらの共有ディレクトリのアクセス制御の設定[10]は構成ファイルと呼ばれるファイルの中で行われる．

1) ノーマル共有アクセス……ノーマル共有アクセスとは，基本的にMCP環境でのファイルアクセス規則がWindows/WindowsNTデスクトップ利用者に適応されるアクセス方法である．共有ディレクトリにアクセスしようとするデスクトップ利用者の利用者コードは，MCPの安全保護（セキュリティ）機構によりチェックされ，その権限が決められる．たとえばMCPのファイルシステムでのプライベートファイルは，MCPの安全保護規則で認められた利用者でなければ見ることはできない．またパブリックファイルは，非特権利用者からも読み出し可能である．この共有ディレクトリ内のファイル名一覧は，MCPのファイル一覧指令を実行したときに表示される内容と同じになる．

OESの構成ファイルには，ノーマル共有アクセスのために次のような登録が必ず行われている．Windows/WindowsNTデスクトップ利用者が，ここで登録されている共有名"_HOME_"をネットワーク接続すると，その利用者コードであたかもMCP制御下のエディタプログラムにログインしたときのようなアクセス環境が設定される．

```
_HOME_          (TYPE       =DISK,
                 PREFIX=    "(<利用者コード>)",
                 FAMILY     ="<ディスクファミリ名>"      )
```

この登録では，_HOME_が共有名である．この共有名は_(下線)で始まっているが，これは特別な意味を持っていて動的に<利用者コード>，<ディスクファミリ名>の部分の置き換えが行われる．<利用者コード>は使用している利用者コードそのもの，またはその利用者コードのMCP制御下での別名利用者コードに置き換えられる．<ディスクファミリ名>は，MCP下の安全保護データベースであ

るUSERDATAファイルに登録されているその利用者コードの一次ディスクファミリ名に置き換えられる．したがってWindows/WindowsNTデスクトップから¥¥<server name>¥_HOME_をネットワーク接続することは，MCP制御下の省略時ディスクファミリの利用者コードのディレクトリに接続することになる．

2) パブリック共有アクセス……このアクセス方法は，MCPの安全保護規則を少し逸脱する．構成ファイルに"PUBLIC=TRUE"と"PREFIX=*PUBLIC"を登録することで，パブリック共有アクセスの共有名になる．この共有名ではMCP制御下の非特権利用者も含め，この共有名で許される全ての利用者がファイルの作成，変更，削除の権限を持つ．

下の共有名PUBFILESの例で説明する．

```
PUBFILES        (TYPE          =DISK,
                 PREFIX        =*PUBLIC,
                 FAMILY        =LANGDV,
                 AREABYTES     =36000,
                 PUBLIC        =TRUE,
                 ACCESS        =ALL
                                -TANSFER -UTIL     )
```

この共有ディレクトリは，TRANSFERとUTILという利用者以外の全ての利用者が使用することができる．MCP管理下の特権利用者，非特権利用者に関係なくLANGDVというファミリで最初のディレクトリノードが*PUBLICというファイルの作成，名前の変更，削除を行うことができる．これ以外についてはMCPの安全保護規則に従う．

3.3.3 CD-ROMサービス

CD-ROMサービスは，ファイルサービスと同様にMCP制御下のCD-ROM装置をネットワークを通して，Windows/WindowsNTデスクトップ環境から使用できるようにする．デスクトップ環境からはCD-ROMは読み取り専用のディスク装置として扱うことができ，複数の利用者が同時に文書ファイル，プログラムを共有することが可能である．

3.3.4 プリントサービス

OESのプリントサービスとは，MCP制御下のプリンタを共有装置としてネットワーク接続されたWindows/WindowsNTデスクトップ利用者に提供することである．これらのプリンタは，MCP制御下のプリントシステムで管理されているので，同時にMCP下のアプリケーションからも共有することが可能である．

3.4 プログラム間通信

プログラム間通信としてOESでは，MCP制御下のアプリケーションとWindows/WindowsNTアプリケーションの間で名前付パイプとWinSockを支援している．Windows/WindowsNT環境では，これらのプログラム間通信機能はよく使われている方式である．したがってデスクトップのプログラム開発者は，これまでと同様に市販のクライアント開発環境，例えばVisual Basic[*5]やPowerBuilder[*6]を使って，OES

とWindows/WindowsNTを組み合わせた分散アプリケーション開発が可能である．

OESでは，これらの二つのプログラム間通信機能をMCP制御下ではポートファイルインタフェースとCOMS（COmmunication Management System）インタフェースと呼ばれる二つの方式で提供している．

1) ポートファイルインタフェース……ポートファイルとは，MCP制御下のアプリケーション間通信方法の一つである．この方法ではアプリケーションは，ディスクファイルをオープンして読み書きするのと同じ様な操作で他のアプリケーションと通信を行う．名前付パイプとWinSockは，MCP制御下ではこのポートファイルインタフェースで提供される．ポートファイルインタフェースでは，名前付パイプ，WinSock両方とも次の五つの文で処理する．

FILE，OPEN，WRITE，READ，CLOSE

名前付パイプの場合，これらはWindows/WindowsNTでは

CreateNamePipe, CreateFile, ConnectNamedPipe,
WriteFile, ReadFile, CloseHandle

に相当する．

したがって，MCP制御下のアプリケーションは，他のMCP制御下のアプリケーションと通信するのと同じ操作でWindows/WindowsNTのアプリケーションと名前付パイプまたはWinSockを使った通信ができる．

また名前付パイプについては，パイプ名と関連するアプリケーション名を構成ファイルと呼ばれるファイルに登録することで，関連するMCP下のアプリケーションの起動を行う仕組みもあわせて提供する．

2) COMSインタフェース……COMSとは，MCP環境下で使われている通信制御ソフトウェアで，オンライントランザクション処理，メッセージの経路制御，ネットワークでの安全保護管理など幅広い機能を持っている．現在MCP制御下のアプリケーションの大部分は，利用者とのインタフェースについてCOMS Windowと呼ばれるインタフェースを使って作られている．COMS Windowには，遠隔ファイルインタフェース，ダイレクトインタフェース，MCS（通信制御ソフトウェア）インタフェースと呼ばれる三つのインタフェース方法があるが，OESでは，MCP制御下のアプリケーションに対し，名前付パイプとWinSockを利用度の高い遠隔ファイルインタフェースとダイレクトインタフェースで提供している．

名前付パイプとWinSockの通信にこの二つのCOMSインタフェースを使うことができることにより，MCP制御下のアプリケーションは，これまで端末装置と通信していたのと同じ方法でWindows/WindowsNTアプリケーションと連携することができる．

3.5 WWW

OESではインターネット（またはイントラネット）環境とのシームレスな接続を実現するため，WindowsNT環境でのWWW（World Wide Web）サーバとは別に，MCP制御下でもWWWサーバを支援している．このWWWサーバでは，MCP制御下のアプリケーションと直接インタフェースするCGI（Common Gateway Inter-

face) ライブラリを提供している.

3.6 1000ユーザを支援する企業サーバ

OES と TCP/IP 接続されたすべての Windows/WindowsNT デスクトップは，チャネルサービスバス経由，802.3，FDDI 経由のいずれの方法においても，MCP 環境で提供するマイクロソフト互換のネットワーキングサービス，たとえばファイルサービス，プリントサービス，名前付パイプなどのすべてを利用することができる．OES の初回のリリースでは，この MCP のマイクロソフト互換ネットワーキングサービスは，ハードウエアの構成により最大で同時に 1000 ユーザを支援する．

このネットワークに接続されたすべての Windows/WindowsNT デスクトップ利用者は，OES のメインフレームの資源，すなわち大規模大量データの管理，データの高い整合性，資源の高度な管理能力を利用することができる．

4. OES の将来

4.1 メインフレームの定義の変化

あるコンピュータが，メインフレームかそうでないかの区別にはいろいろな考え方があるが，従来言われていた固有性は，マイクロソフトの独占によってあまり意味を持たなくなってきている．メインフレームの定義は，プラットフォームというより，むしろ，「高処理能力用途の問題解決能力が備わっているかどうか」に変わってきていると考えられる．例えば，グラフィカルユーザインタフェース以外の部分はダム端末モデルである X アプリケーションしか動かさない UNIX であっても，それを「メインフレーム」とは呼ばないのはユーザにそういった意識があるからだろう．また，A シリーズのハードウェアインタフェースやオペレーティングシステムを，PC プラットフォームで完全にエミュレーションする技術も既に現実化しており，もはや従来式のメインフレーム・非メインフレームの分類は意味を持たなくなってきている．

逆に，最近の PC サーバは，基盤となるハードウェアやソフトウェア面でメインフレームが得意としていたところの機能（表 1）を取り込みつつある．

表 1 以前の PC，メインフレームの特性

PC が得意とする分野	メインフレームが優っている（た）分野
・グラフィカルユーザインタフェース エンドユーザの好みの反映 エンドユーザプログラミング ・分散処理 ・カスタマイズ可能な安価なコンポーネント ・使い易いシステム管理ツール	・複数ユーザの管理 ・入出力基本性能と入出力最適化 ・スケーラビリティ 信頼性 可用性 ・セキュリティ

ただ，如何に多くの PC を並列にしても，ネットワークバンド幅や管理コストの問題を吸収するには至らない，すなわち単純にはメインフレームの代替にはなり得ないことにユーザは気が付いてきており，PC サーバ機については頑強性を増す方向にある．もはやハードウェア単体でメインフレーム機能を遂行することは非現実的であり，企業を維持するための様々な機能が異なるプラットフォームで遂行されると考えるべき

であろう．

つまり，PCベンダによるPCの強化と今回，OESが採用したHMP結合は，仮想的なメインフレームの構築，「新しい概念によるメインフレーム再構成」，を目標にしているといえる．

4.2 コンピュータの仮想化

もう一つの大きな流れは，コンピュータの仮想化である．OESはWWWサーバとして機能することにより，この分野への適用性があると考える．インターネット（ないしイントラネット）においてLoad & Executeスタイルの実装が進むにつれ，これが一つの大きなコンピュータのようであることが最近指摘されるようになってきている．具体的にはJavaやGeneral Magic社のTelescriptは，仮想マシンを定義し，そのための仮想コードを実行するようになっている[11][12]．これらに共通しているのは，徹底的に抽象化されたコンピュータである．Load & Executeという意味では，PostScriptも同じであるが，これは仮想コードではなく，物理的な動作にダイレクトにマッピングされているので，世代が異なると考えるべきであろう．例えばNetscape Navigatorは，そこにコンテンツを実装する立場から見れば（アプリケーションではあるが）一種のOSのように見える．一方，Javaのアプレット（JDK：Java Development Kitを用いて生成されたコードとデータ）や，Telescriptは自律性を持ち，自身で移動（travel）が可能である．これは，一般的なRPC（Remote Procesdure Call）手法だけでは実装することはできない．同期型のRPCは確かに利にかなった技術ではあるけれども，全てに同期性を要求する部分と制御モデルに一定の規則（かならず制御がOriginatorに戻らねばならない等）が制限事項となっている．例えば同期性については，光ファイバを占有できたとしても，8KBのデータを送るのに，8KBのウィンドウサイズで東京・大阪の往復に約6.7ミリ秒かかるといわれており[13]，通信の多重度やTCP/IPプロトコルの特性（パケットヘッダによる通信ロス等）を考えると，全てをRPCでという考え方で可能になるかどうかはわからない．UNIXでよく使用される手続きを呼び出すプロセスが，ブロックされない非同期RPCについても，RPCという形態をとる限り制御の形態が固定的であり，結果的にエージェント指向におけるリモートプログラミング（RP）の概念も合わせて必要になると考えられる．総合的にはこれらは図5のようなモデルで示される．

このような考え方は，RPCを用いたクライアントサーバモデルのスーパセットと位置付けることができるが[14]，優劣の問題ではなく解決すべき問題の解法の選択という棲み分けの問題である．ここでいう「場所」は物理的なものではなく，一企業全体であるかもしれないし，単体のコンピュータであるかもしれない．

4.3 分散オブジェクトへ

これまでに述べた「仮想的なメインフレームの構築」と「コンピュータの仮想化」を支える技術的基盤は，「RPを包含した分散オブジェクト」であると考えられる．例えばWWW（World Wide Web）におけるURL（Uniform Resource Locator）は，オブジェクト参照の一種であると考えられる．

図6のようなモデルを具現化するためには，単純なRPCやエージェント指向関連技術を実装するというより，複数の選択枝から選ぶことになる．そこではOSF(Open

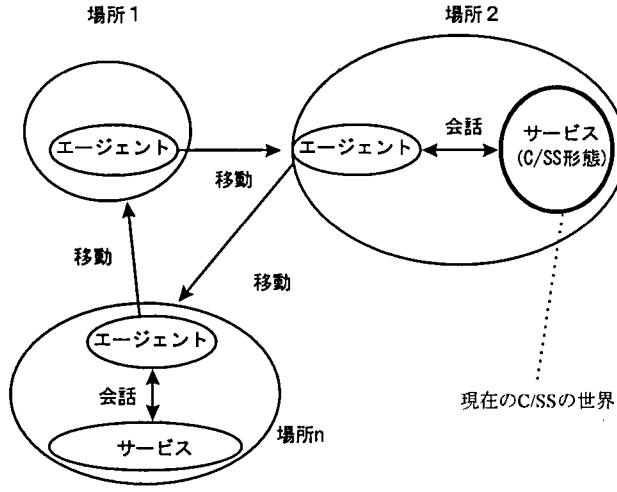


図 5　エージェント型クライアント・サーバモデル

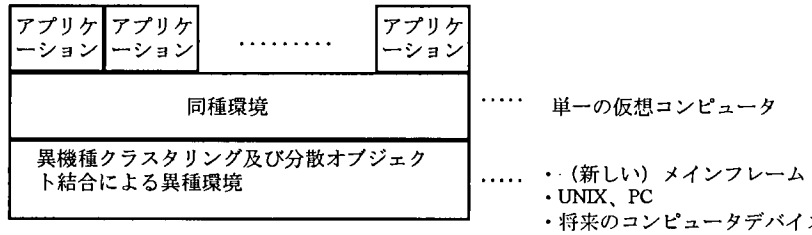


図 6　コンピュータの仮想化

Software Foundation）の DCE（Distributed Computing Environment），OMG における CORBA（Common Object Request Broker Architecture）やマイクロソフト社の Distributed COM（Component Object Model)/Distributed OLE などがベースになると考えられる．

メインフレームの概念そのものが変化した後では，分散オブジェクトへ対応できないコンピュータは，現在のメインフレームであれ UNIX であれ PC であれ，適用が困難になるであろう．OES ではマイクロソフトが実装する Distributed COM ベースの分散 OLE の恩恵を受ける事ができるため[15]，その意味で新しいメインフレームへのパラダイムシフトに対応できると考えられる．また，CORBA についても同様のことが言える．

ただ，オブジェクト指向技術全体（オブジェクト指向プログラミング，データベース，システム設計，オペレーティングシステム）は 1990 年代に入ってから，その重要性が急激に認識されつつある．PC においては MFC（Microsoft Foundation Class）クラスライブラリ（ないしその競合品）から派生クラスを生成して部品を作成する層と，それを組み合わせて使う層の 2 極化が進んでいる．またその間は Visual Basic に代表される RAD ツールがカバーしている．

この実装自身は OES ないし A シリーズにとって難しい事ではない．むしろ，Tele-

scriptなどが，機密保護の面からのメモリアクセス制限やアルゴリズム記述における保護ブロックの実装など「特長」としてうたっている点は，本論文集の「Aシリーズオペレーティング・システムの発展」にも述べられているとおり，Aシリーズでは既にアーキテクチャレベルで実装済みであり，トロイの木馬に代表されるウィルスの進入など不可能である．

むしろOESのマイクロソフトアプローチが功を奏するかどうかは，プログラミング技法ばかりではなく，ビジネスオブジェクトのクラスライブラリ化における各種の技法（オブジェクトのモデリング技法における動的モデル，機能モデル，オブジェクトモデルなど）への習熟やモデリングの際の視点を養うといったような，メインフレームに関わってきた人間が，この文化にどれくらい近づけるかという，使う側のパラダイムシフトにかかっているといえる．

5. おわりに

HMP結合を行うOESにとっては，PCの急激な台頭は，歓迎されるべきことである．重要なのは異種の結合ということであり，Windows/Windows NTの結合ということではない．今回の実装により，将来別のものを結合し，クラスタリングすることも可能である．

HMPのような新しい概念のメインフレームを，ミッションクリティカルな業務分野に適用する技術はまだそれほど確立しているわけではない．そしてそれは分散オブジェクトに対応してゆくこれからの数年にかかっているといえる．新しいメインフレームが完全な「オブジェクト」として従来のアプリケーションに維持されるデータを格納，適合（カプセル化）できるように，クラスタネットワークないしインターネットでふるまう事ができた時，はじめてそれが「パラダイムシフト」という意味を持つのだと考える．さもなくば，メインフレームは融通の効かない巨大な倉庫となるであろう．OESは，このような世界にメインフレームを適合させていく製品である．

＊1 ODBC, WindowsNTはMicrosoft社の商標，Windowsは登録商標である．
＊2 UNIXはX/Openカンパニーリミッテッドがライスセンスしている米国ならびに他の国における登録商標である．
＊3 Netwareは Nobell社の登録商標である．
＊4 Post Scriptは Adobe Systems社の登録商標である．
＊5 VisualBasicは Microsoft社の登録商標である．
＊6 PowerBuilderは Powersoft社の商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

参考文献
［1］「ダウンサイジングの実施内容（業種別）」情報化白書95, 1995年5月31日, 財団法人日本情報処理開発協会,（株）コンピュータ・エージ社.
［2］「MAINFRAME TO WEB WITH WEB OBJECTS」http：//www.next/com/WebObjects/Mainframe_Brief.html Feb, 1996. NeXT software Inc.
［3］ Andy Reinhardt,「YOUR NEXT MAINFRAME」BYTE, May 1995, pp, 48～58, McGraw-Hill.
［4］ Johnson M. Hart & Barry Rosenberg,「Standard and Open Systems」, Client/Server Computing for technical professionals, Addison Wesley.
［5］ 近藤仁志,「並列処理システムの分類」インターフェース, 1993年2月, pp, 98～115, CQ出版社
［6］ 清水・堀江・石畑,「高速メッセージハンドリング機構」1993年4月, 情報処理学会論文誌 Vol. 34 No. 4,（財）情報処理学会.

堀江・小柳・今村・林・清水・石畑,「メッセージ通信の分散メモリ型並列計算機性能への影響」1994年4月. 情報処理学会論文誌 Vol. 35 No. 4, (財)情報処理学会.
[7] John D. Ruley,「Programming NT Networks」Networking Windows NT, John Wiley & Sons, Inc.
[8] X/Open CAE Specification Protocols for X/Open PC Interworking: SMB, Version 2, X/Open Company Limited.
[9] X/Open CAE Specification IPC Mechanisms for SMB, X/Open Company Limited.
[10] Functional Design Specification WinServices Functional Design, Unisys Corporation.
[11]「The Telescript Programming Guide. Version 1.0 Alpha」October 1995, General Magic, Inc. 420 North Mary Avenue Sunnyvale, CA 94086.
[12] James Gosling & Henry McGilton,「The Java(tm) Language Environment: A White Paper」1995, Sun Microsystems.
[13] 天海良治「さすがのネットも光速を越えられない」Bit 1996 Vol. 28, No. 1 共立出版.
[14] Anshley McClenaghan,「Script and Mobile Agents Briefing Notes」, September 1995, Architecture Project Management Limited (APM).
[15] Sara Williams & Charlie Kindel,「The Component Object Model」INTEROPERABLE OBJECTS REVOLUTION, Dr. Dobb's special report, Mar 1995.

執筆者紹介 室　木　　通 (Toru Muroki)
1950年生. 1972年上智大学理工学部電気電子工学科卒業. 同年日本ユニシス(株)入社. Aシリーズのオープン関連プロダクトの受入評価, 日本化, 保守に従事. 現在, システムプロダクト部OES開発推進室に所属. E-Mail: Toru.Muroki@unisys.co.jp

原　　広　仁 (Hirohito Hara)
1958年生. 1981年京都産業大学経営学部経営学科卒業. 1981年日本ユニシス(株)入社. Aシリーズプロダクトの利用技術支援に従事. 現在, 関西支社システム技術室に所属. E-Mail:Hirohito.Hara@unisys.co.jp

シリーズ2200オペレーティング・システムの発展

The Evolution of 2200 Series Operating System

伊藤龍彦

要約 ユニシスの大型メインフレームであるシリーズ2200のオペレーティング・システム（OS 1100）の起源は，UNIVAC 1107システム（1962年）のオペレーティング・システムであったEXEC Iである．それから本年，1996年まで34年間のOS 1100の歴史はコンピュータそのものの歴史に等しい．OS 1100は単に生き長らえたのではない．現在のOS 1100は最近開発されたオペレーティング・システムに伍し，優位性を保っている．この事実は，当初のOSの基本設計の優秀さと，その後の保守・改善の適切さを証明している．本稿ではOS 1100の誕生から現在までの発展をたどる．

Abstract OS 1100, the operating system for 2200 Series systems, Unisys large scale mainframes originates from EXEC I operating system running on UNIVAC 1107 system (circa 1962). Since then till this year (1996), thirty-four years' history of OS1100 has reflected the history of computers themselves. OS1100 has not only just survived, but still ranks with or, in some cases, maintains a stronger position over the recently developed operating systems. This fact shows the superiority of the original base design and the appropriateness of later maintenance and enhancement of this operating system. This paper trances the history of OS1100 from its birth to the present day.

1. はじめに

シリーズ1100の名称の由来は，1950年のUNIVAC 1101にまで遡る．UNIVAC 1101は，ENIAC（1945年）の5年後，世界で13番目（2進数で1101）のコンピュータとして開発され，UNIVAC 1101と命名され科学技術計算を中心に利用された．その後，UNIVAC 1102，UNIVAC 1103（自動プログラム割込の採用），UNIVAC 1105を経て，1962年のUNIVAC 1107に至る．UNIVAC 1107システムにおいてシリーズ1100（すなわちシリーズ2200）の基本的なアーキテクチャが確立された．シリーズ1100の最初のシステムは1107である．

1107システムが世に出たのは，今から34年前の1962年であった．歴史年表を見れば，それが60年安保の2年後，東京オリンピックの2年前であることが知れる．1962年の1107から，1996年1月に発表された，シリーズ1100アーキテクチャを継ぐ最新システムであるITASCA 3800シリーズまで，実に34年が経っている．その間のコンピュータ関連技術の発展には，ハードウェアにおいてもソフトウェアにおいても，目覚ましいものがあった．価格低下や性能向上は当時の想像を超えて進んだ．パーソナル・コンピュータは今や生活に欠くことの出来ない道具となろうとしており，人間とコンピュータの関係そのものが変ってしまった．インターネットに代表されるコンピュータ・ネットワークの発展は，文化に著しい影響を与えるようになった．この34年間に，シリーズの呼称も，UNIVACシリーズ1100からUnisysシリーズ2200に変わり，オペレーティング・システムの呼称も，EXEC I，EXEC II，EXEC 8，EXEC 8 E，

OS 1100 へと変わった.

この 34 年間，シリーズ 1100･2200 は，メインフレームに対する期待の変化に応え，また，技術の発展に対応して革新的な改良を重ねたにもかかわらず，基本的なアーキテクチャは不変で，唯一のオペレーティング・システムを維持してきた．このことがシリーズ 1100 の基本設計の優秀さとその後の改良のポリシーの正当性を証明している.

本稿では，シリーズ 2200 のオペレーティング･システムの系譜をたどり，基本設計と改良のポリシーとを概観する．系譜をたどれば，シリーズ 2200 とそのオペレーティング・システムの進むべき道筋がおのずと示唆されるのではないか.

2. OS 1100 の系譜

2.1 1107 システムとそのオペレーティング・システム

シリーズ 1100 (すなわちシリーズ 2200) の基本的なアーキテクチャが確立されたのは UNIVAC 1107 システムである．その主な特徴は次の通りである.

- 36 ビット語
- 1 の補数による演算
- 16 個のインデックス・レジスタと 16 個の演算レジスタ
- メモリとレジスタのオペランド間の演算（二つのレジスタのオペランド間の演算も可)
- 間接アドレス付け
- 16 ビットアドレス領域とインデックスによる絶対アドレス付け（最大メモリ 65 K 語（65,536 語))

シリーズ 1100 のオペレーティング・システムは EXECUTIVE，あるいは単に EXEC と呼ばれた．最初の EXEC である EXEC I は，1962 年に 1107 用にリリースされた．EXEC I はタイム･カンタムによるプログラムの制御の切り替え機構 (多重プログラミング) を提供し，さらに現在の EXEC の機能のうち多くの機能を提供した，すなわち，

- 優先度に基づくジョブ・スケジューリング，
- 装置の論理的割り付け，
- 相対アドレスに基づくプログラムの実行，
- 多重プログラミング

である．一方，EXEC I では提供されなかった主な機能は，ファイル管理機能および簡易な制御言語であった．これらの機能不足を解決するためには，EXEC を書き直さなければならなかった.

1963 年初めに 2 番目の EXEC である EXEC II がリリースされた．EXEC II は 1107 の主要な EXEC となった（EXEC I と EXEC II とは，初期の 1108 でも使用された)．EXEC II は EXEC I と並行して開発されたが，初期の EXEC I での経験を活かして，柔軟で使い易い制御言語（EXEC 制御文）をもち，プログラム･ファイル/データ･ファイル機能を持っていた．また，EXEC II にはシンビオントと呼ばれる入出力制御ルーチンの概念が導入された．シンビオントによって，プログラムの実行と同時に紙テ

ープやカードの読み込み，プリント，パンチが行えるようになった．しかし，EXEC I では提供されていたユーザ・プログラムの多重プログラミング機能は未だ提供されていなかった．各プログラムは順次に実行された．

1107 のアドレス方式の特徴は，次の通りである．

1107 シリーズの演算用のレジスタは，ゼネラル・レジスタ・セット（General Register Set, GRS）と呼ばれ，主記憶装置のアドレス 0_8〜200_8（8 進数表記）と重複している．演算はほとんどこれら高速のレジスタ間で行われる．1107 の命令語の形式を図 1 に示す．二つの 4 ビットの指示部，a および x，が GRS の 3 種類のレジスタのいずれかを指示する．GRS は，指示用の X レジスタ，計算用の A レジスタ，およびリピート回数を示す R レジスタの 3 種からなる．X レジスタは図 2 の形式である．命令語の x 指示部が指定されていない場合(0 の場合)，オペランドのアドレスは，命令語の u 指示部（16 ビット）が示す値である．u 指示部の値が 200_8 未満の場合，レジスタを指示するとみなされ，レジスタ間の演算が行われる．命令語の x 指示部が指定されている場合 (0 でない場合)，オペランドのアドレスは，u 指示部の値と指示された X レジスタのモディファイア部分（18 ビット）の値との和の値である（$u+X_m$）．

h 指示部が 1 の場合，X_m の値を X_l 分増加する演算（$X_l+X_m \rightarrow X_m$）が行われ，配列をオペランドとする演算が簡易になる．

a 指示部 (4 ビット) は，もう一方のオペランドとなる演算レジスタを指示する．通常は A レジスタが指示されるが，X レジスタや R レジスタであってもよい．

f 指示部 (6 ビット) に指示された演算は，a 指示部で指示された A レジスタと u 指示部（および x 指示部）で指示されたアドレスとをオペランドとして実行される．j 指示部(4 ビット)は演算の種類を示すために補助的に使用されるが，幾つかの命令では，演算の対象とするオペランドの部分 (1/2 語 (18 ビット)，1/3 語 (12 ビット)，1/6 語 (6 ビット)) を示すのにも使用される．

1107 でのアドレス計算例を図 3 に示す．

2.2 UNIVAC 1108 システムと EXEC 8

EXEC I と EXEC II は多くの機能を提供したが，共通の欠点があった．それはその基本設計がバッチに基づいていたことだった．時あたかもタイムシェアリングが登場

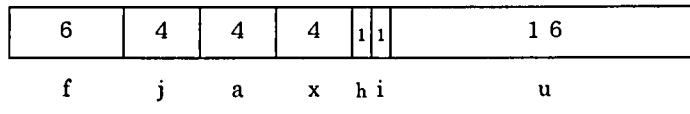


図 1　1107 の命令語の形式

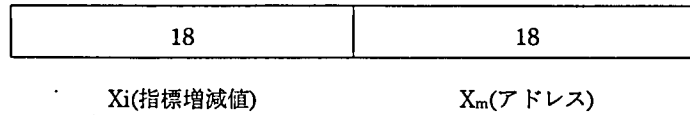


図 2　指標レジスタの形式

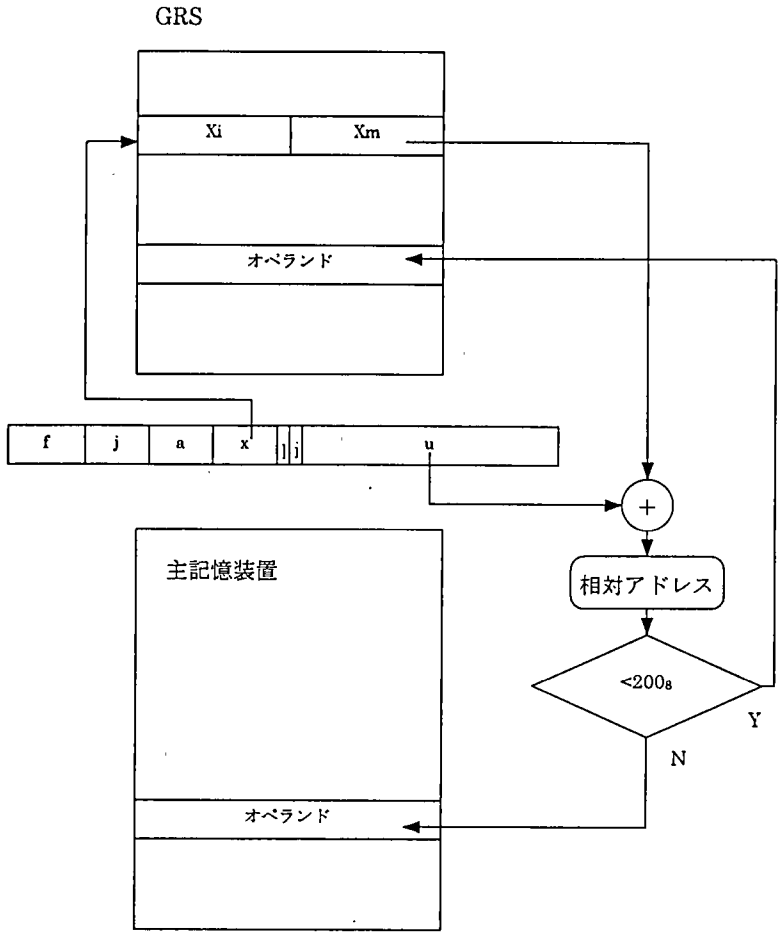


図 3　1107 のアドレス計算例

しようとしていた．タイムシェアリングに対応するには，EXEC を再度書き直さなければならなかった．そして，UNIVAC 1108 システムの導入を機に，全く新しい EXEC を開発する決断がなされた．

新しい EXEC は，EXEC 8 と呼ばれ，1967 年，1108 用に初めてリリースされた．それは EXEC I の多重プログラミング機能と EXEC II の EXEC 制御文，プログラム・ファイル/データ・ファイルおよびシンビオント機能を統合したものであった．EXEC 8 は，多重プログラミングおよび多重プロッセッサ環境で稼働するように設計され，バッチ，遠隔バッチ，デマンド，および，トランザクションの各形態のプログラムを同時に動かすことができた．とくにデマンド（タイムシェアリング）が強化された．バッチ・モードで使用できるシステムの資産は全てデマンド・モードでも使用できる．一つのラン・ストリームが変更なしにバッチでもデマンドでも使用できる．EXEC 8 にはトランザクション・モードもある．これは航空会社の座席予約システムや銀行の出納システムのようなリアルタイム環境で使用される．

以下に，EXEC 8 で使われる幾つかの重要な用語およびシステム概念を定義する．

「ラン・ストリーム」は，ユーザ・サービスへの要求の集合であり，バッチ，遠隔バッチ，および，デマンドで使用される．ラン・ストリームには各要求に対するデータも含まれる．ラン・ストリームはEXEC制御文で記述される．ラン・ストリーム中の各要求，たとえば磁気テープ装置の割付要求，プログラムのコンパイルの要求等を，「タスク」と呼ぶ．ラン・ストリームの実行のことを単に「ラン」と呼ぶ．

プログラムはラン・ストリーム中の「シンボリック・エレメント」として入力され，コンパイルされて，相対アドレスに基づく「リロケータブル・エレメント」に作り直される．これがライブラリのリロケータブル・エレメントと共に集められて，一つの「アブソルート・エレメント」，すなわちプログラムになる．ただし，アブソルート(絶対) なのは，プログラム内部のアドレス付けであって，ハードウェアのアドレスとしては相対的なままである．プログラムがメモリにロードされた時に初めてハードウェアとしての絶対アドレスが決まる．

バッチのラン・ストリームがシステムに入力されると，まずシンビオントで処理される．実行に必要な機器を割付るために，ラン・ストリーム中の要求が検査される．シンビオントでは多重で非同期な入出力処理が可能であるが，これは多重プログラミング/多重処理環境ではとくに重要な機能である．

ラン・ストリームは大記憶装置の待ち行列の後に，コース・スケジューラによってスケジュールされ，選択され，実行が開始される．ランには一時的なプログラム・ファイルが割り付けられ，コンパイラやテキスト・エディタの出力がラン・ストリームの実行中，一時的に保存される．

EXEC 8は，1967年の初期リリース後，新しく導入されたシステム・ソフトウェアの通例として，安定性の急速な向上と比較的小さな機能追加とを行った．最も重視されたのはデマンド環境でのEXEC 8の問題の修復であった．問題になったのはバッチ・プログラムの実行の繰り返しではなく，多数のデマンド端末のセッションの運用であった．デマンド・プログラム間の資源の配分を制御するダイナミック・アロケータ，ディスパッチャと呼ばれるルーチンの資源配分のアルゴリズムのチューニングに多大な労力が費やされた．もともとのタイムシェアリングのアルゴリズムはプロセッサ使用にのみ基づいており，これでは実際には頻繁なプログラム・スワップを繰り返し，不十分であることがわかった．改良されたアルゴリズムでは，各プログラムに配分される時間の決定に，入出力およびプログラム・サイズを考慮に入れている．

このような産みの苦しみにもかかわらず，EXEC 8は強固な設計であることが証明された．永年にわたって多くのルーチンやアルゴリズムが書き直され，多くの新しい周辺機器の処理ルーチンが追加された．しかし，基本設計は変更されていない．これには「プログラマ・リファレンス・マニュアル」の存在によるところが大きい．開発が開始される前にプログラマ・リファレンス・マニュアルが作られ，それが設計や開発を制御する道具として使用された．

1108に続くシリーズ1100の導入はEXEC 8の基本ロジックの試練となった．入出力制御，プログラム・スワッピング，インタラプト処理等，時間に関係するルーチンは，比較的処理速度の遅い機種，1106等から1110や1100/80等の超高速機種に対応して調整された．

「EXEC 8 の全構成要素のソース・プログラムを唯一つだけ保持する」という基本設計の目標が全システムに同一の環境を提供するための要点であることが証明された．シンボリック・エレメントは全てのサイトに提供された．システム生成ツールが提供され，サイトで特定の機能を組み入れる，あるいは取り除くことができる．EXEC 8 の場合，これらのツールはモデルに依存するコードを組み込み，あるいは，取り除けられるように拡張された．特定のモデルでしか提供されない命令語は極力隔離するように注意が払われた．このようにして，同じ EXEC8 の同一のシンボリック・バージョンが 1107 以降の種々の 1100 システムで EXEC 8 を生成するのに使用できる．種々の 1100 システムのアーキテクチャのかなり大きな違いにもかかわらず，EXEC 8 のハードウェアに依存する部分はわずか 10 %程度にすぎない．新しい EXEC を開発したり，複数の EXEC を保守する巨額の費用を考慮すると，単一シンボリックの概念なしにシリーズ 1100 の製品ラインが今日あるように多様であり得たかどうか疑わしい．

EXEC 8 は，コンパイラやデータベース管理システムのようなシステム・ソフトウェア，およびユーザ・プログラムをコンピュータの物理的性格から「絶縁」している．たとえばファイル管理において，ファイルはシンボリックな名前がつけられ，論理的な機器のクラスに割り当てられる．ファイルは，プログラムの変更なしに，磁気ドラムから固定ディスクに，リムーバブル・ディスクに移すことができる．ファイルが長期間にわたって使用されないと，EXEC 8 はそのファイルをテープに移し，次に参照されるときにマス・ストレージに戻す．このような仮想ファイル構造内でのファイルの動きは，多少時間がかかることを除けば，ユーザには直接のかかわりがない．各ファイルは論理的なアドレス付けされた空間とみなされる．プログラム・ファイルのシンボリック・エレメント，リロケータブル・エレメント，アブソリュート・エレメントのユーザアクセスはすべてエレメント名で行われ，エレメントの論理アドレスは参照されない．

1108 のアドレス方式の特徴は次の通りである．

1108 システムのアドレスは相対アドレス方式が可能となった．プログラムを命令部分とデータ部分（これをそれぞれ「バンク」と呼ぶ）に区分し，各バンクを主記憶装置（メモリ）のどこにでも動的に配置できるようになった．1108 のハードウェアでは二つの基底レジスタを導入することで動的再配置が可能となった．1108 のアドレス計算を図示する（図 4）．

1107 では，$u+X_m$ はオペランドの実アドレスであるが，1108 の場合は一般的に，バンク内の相対アドレスである．相対アドレスの場合の実アドレスの値は，相対アドレスに選択された基底レジスタの値（基底値）を加算した値，すなわち，$u+X_m$＋基底値となる．

基底レジスタの長さは 18 ビットなので，262 K 語の記憶領域が使用可能である．通常，この領域は EXEC と多重プログラム化された幾つかのプログラムによって占められる．各プログラムは指標付けなしでは 65 K 語，指標付けられた場合は，X_m が 18 ビット長であるから，262 K 語のデータ領域をアクセスできる．

記憶領域は 512 語の境界で割り付けられ，保護される．セグメント外へのアクセスは禁じられており，セグメント内での書込み保護が選択できる．

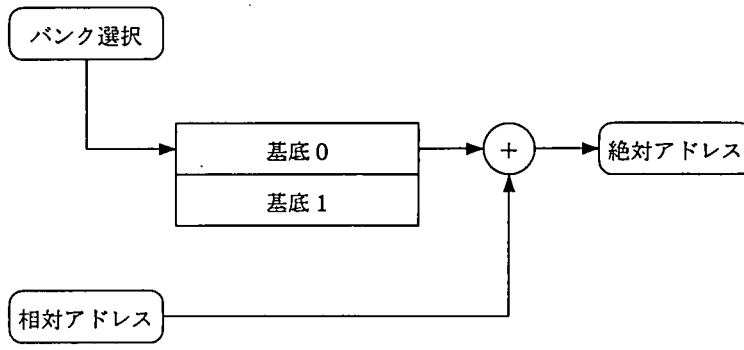


図4　1108のアドレス計算例

1108の相対アドレス方式による再配置機能によって，効果的な多重プログラミングが可能となった．特権モードとメモリ保護機能によって多重プログラミングに必要な隔離が可能となった．システム資源の制御命令，たとえば入出力命令は特権に区分される．多重プログラミングにはGRSの二重化が必要になる．ユーザ用のGRSとEXEC用のGRSとである．ユーザはEXEC用のGRSを使用できないが，EXECは両方を使用できる．二重化によってGRSを状況によって切り替える時間が節約できる．切り替え時間をさらに節約するために，EXECがユーザGRSを使用するときはレジスタのマイナーセットを使用するようなソフトウェア上の規約が設けられている．

2.3　1110システムとEXEC 8E

1972年の1110システムは，1108システムの多重プログラミング機能，多重プロセッシング機能等を強化することによって，拡大・進化するビジネス環境に対応するシステムであった．EXEC 8は大きく改良され，多重バンクによる仮想記憶，QUOTAシステムによる自動資源管理，端末保証システム（TSS），テープ・ラベル・システム（TLS），SUP（Standard Unit of Processing）概念によるコンピュータ使用時間の計測等の機能が提供された．改良されたEXEC 8は，EXEC 8 Eと呼ばれた．

1110のアドレス方式の特徴は次の通りである．

1110では4個の「バンク記述子レジスタ」を導入し，これらに，現在アクティブなバンクのバンク記述子を最大4個まで格納する方式が採用された．また，4,096個のバンク記述子を収める「バンク記述子テーブル」を導入した．ユーザはバンク記述子レジスタにバンク記述子テーブルからバンク記述子を設定する．バンク記述子テーブルには4，096個のバンク記述子があり，各バンクは最大262 K語の長さなので，ユーザが4個のバンク記述子レジスタに，次々にバンク記述子テーブルからバンク記述子を設定すれば，計算上，合計約10億語（262 K語×4096）のアドレス空間を使用できることになる．

1110のアドレス方式の概念図を図5に示す．

一つのバンク記述子がバンク記述子テーブルの複数の位置に現われ得る．これは複数のユーザが一つのコードを共有できることを意味する．これによって主記憶装置のスペースを節約できる．

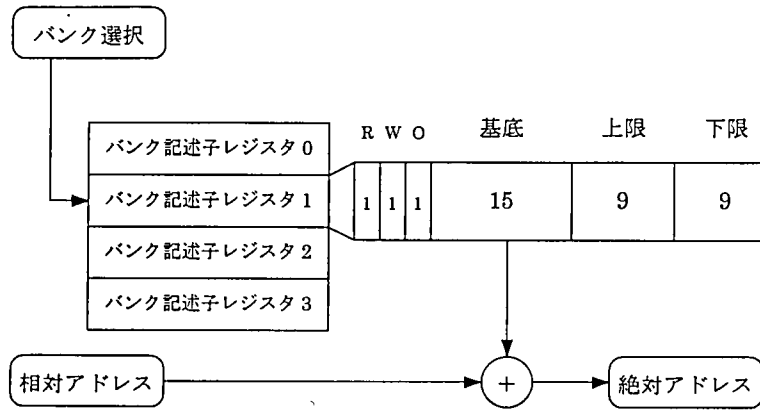


図 5　1110 のアドレス計算例

1110 では，基底値は 18 ビットから 24 ビットに増加し，より大きな記憶領域がアクセス可能になった．巨大な記憶領域の適切な価格を維持するために，記憶領域は，262 K 語の高速高価格の主記憶領域と 1 M 語の低速低価格の拡張記憶領域との 2 レベルとすることになった．

2.4　1100/80 システムと OS 1100

1976 年の 1100/80 システムで，シリーズ 1100 の名称が 1100/xx シリーズと変更され，EXEC 8 E も「OS 1100」と名称が変更された．EXEC と呼ばれた時代，EXEC はコンパイラやデータベース管理システム等のソフトウェアを含まない狭義のオペレーティング・システムを指す意味で使用されていたが，「OS 1100」は，これ等のソフトウェアを含む広義のオペレーティング・システムを指す名称となった．

1100/80 のアドレス構造は 1110 に似ている．相違点は，バンク選択のアルゴリズムを単純にしたことである．また，バンク保護が木目細かくなった．ユーザは，512 K 語の境界から開始して，64 語の境界で終わり，最大 262 K 語の大きさのバンクを指定できる．バンクの保護もこの単位で行われる．

2.5　1100/90 システムと OS 1100

1982 年の 1100/90 システムは，オフィス・オートメーション（OA）の普及による分散処理の急激な進展と情報処理の量の拡大に対応し，使い易く安全なコンピューティング環境を提供するため，従来のアーキテクチャに追加して，新たに「拡張アーキテクチャ」を提供した．拡張アーキテクチャでは，巨大仮想アドレス空間（69 ギガ語）の提供，リンク/ドメインによる保護機構等の新機能が提供された．これらの新機能は，新しいアドレス方式と新しい形式の命令語によって実現されている．拡張アーキテクチャでは，既存のプログラムはそのままでは実行できない．既存のプログラムを 1100/90 シリーズで稼働させるために，OS 1100 には，従来の 1100 と互換性を保つための「基本モード」，1100/90（およびそれ以降のシステム）の新機能を使用できる「拡張モード」，および基本モードのプログラムと拡張モードのプログラムとの間で相互アクセスをするための「共存モード」の三つのモードが設けられている．

新形式の命令語，新しい拡張モードの導入と共に，実行用プログラムにも新しい形

式，目的モジュール（Object Module），が導入された．シンボリック・エレメントがコンパイルされ，直接，目的モジュールが生成される．目的モジュールは，そのまま実行可能である．ライブラリ等への外部参照は，リンキング・システムが実行時に解決する．リンク・プロセッサを使用して，外部参照をあらかじめ解決させることもできる．目的モジュールは，バッチ，デマンド，トランザクションのいずれの形態でも使用できる．

これらの新機能にもかかわらず，シリーズ1100およびOS 1100としての特徴は，むしろ，従来のプログラムが何ら変更もなしに，1100/90シリーズで稼働すること（互換性）にこそある．これら三つのモードを使い分けることによって，ユーザは優先度の高いものから徐々に新しい拡張モードに移行することが可能になる．

1100/90シリーズの新アドレス空間の導入における方針は次の通りである．

- アドレス空間を拡張する．
- 既存のユーザ・ソフトウェアの互換性を将来にわたって保証する．
- このために，従来のバンク方式を拡張した形で新アドレス方式を導入する．

このため，1100/90の論理アドレス構造は，1100/80の論理アドレス構造を継承し，さらに次のような改良を加えた．

- 1100/80の4個のバンク記述レジスタを廃し，基底レジスタ（Bレジスタ，ユーザ用16個およびシステム制御用16個，各72ビット）を新設した．
- バンク記述子テーブル（BDT）を拡張し，バンク記述子（BD）を最大65,536（$=2^{16}$）個収容できるようにした．BDTの個数はシステムに四つまでに拡張された．
- 各バンク記述子（BD）を4語に拡張し，262K語のバンクごとに一つのバンク記述子を対応させた．
- プログラム・アドレス・レジスタ（PSR）の形式を，24ビット実アドレスから36ビット仮想アドレスに変更した．36ビット仮想アドレスであるプログラム・アドレス空間は，最左端の2ビットがレベル指示部でBDTを指示し，次の16ビットがバンク記述子インデックス（BDI）で指示されたBDT内で使用するバンク記述子（BD）を指示し，残りの18ビットがオフセットで，そのバンク内でのオフセットを指示する（図6）．

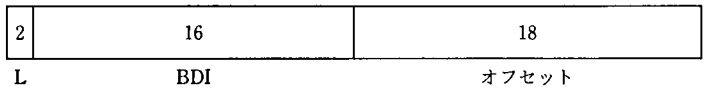


図6　1100/90の36ビット仮想アドレス

- 従来の命令語の形式に加えて，新しい命令語の形式を導入した．新形式の命令語では，u指示部（16ビット）を，Bレジスタを指示するb指示部（4ビット）とオペランド指示のためのd指示部（14ビット）に分割した（図7）．
- 指標レジスタの形式が変更された（図8）．

1100/90においてプログラムのアドレス空間は69ギガ語（68,719,476,736語，275ギガバイト）に拡張された．1100/90システムでのアドレス計算例を図9に示す．

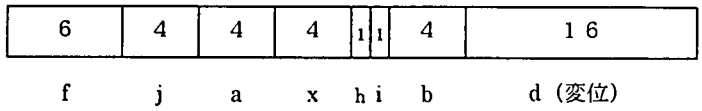


図 7　1100/90 の命令語の形式

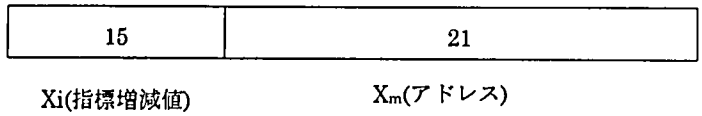


図 8　指標レジスタの形式

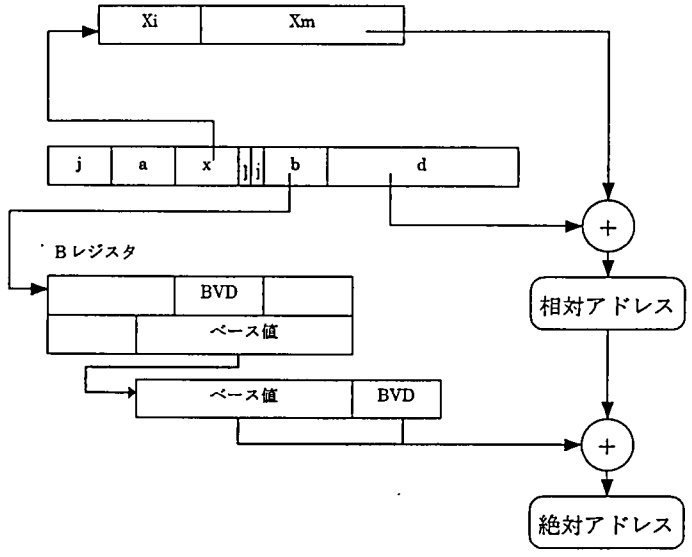


図 9　1100/90 のアドレス計算例

1100/90 では，プログラムの実行に，三つのモードがある．

- 基本（ベーシック）モード：従来の 1100 シリーズで使用してきたソフトウェアと互換性を保つために用意されたもので，従来のプログラムの命令語の形式がそのままの形で実行される．
- 拡張（エクステンデッド）モード：拡張された仮想アドレス空間を使用するために用意されたもので，新形式の命令語が実行され，36 ビットのアドレス空間が利用できる．
- 共存（ミックス）モード：基本モードの命令語からなるプログラムと，拡張モードの命令語からなるプログラムとの間で相互にアクセスできるように用意されたモード．

拡張モードのプログラムを開発するためのコンパイラ群として，UCS（Universal Compiling System，汎用コンパイル系）が用意された．プログラミング言語として，従来の COBOL，FORTRAN，および PLUS に加え，新たに Pascal および C が提供

されている.

2.6 シリーズ2200とOS 1100

1986年のシリーズ2200/200は,メインフレームとして最初にCMOS VLSIを採用した点で画期的なコンピュータであった.シリーズ2200という名称でもって,新世代のコンピュータの登場を明示した.その後,CMOS VLSI化は,中型機から大型機の2200/400シリーズ(1988年),2200/500シリーズ(1993年),および超大型機のITASCA 3800シリーズ(1996年)へと進む.

1980年代後半になると,ハードウェア技術の進展以上に,コンピュータ世界のパラダイム・シフトが起こる.パーソナル・コンピューティング,ネットワーク・コンピューティング,オープン化,クライアント/サーバ・システム等が出現し,汎用機の位置付けが変化し始める.シリーズ2200においても,1991年の超大型機2200/900シリーズは,前年に発表した,オープン・システムを取り入れた新ソフトウェア・アーキテクチャ“Unisys Architecture”(UA)の情報中枢(Information Hub)として位置付けられた.さらに,オープンとの共存・協調を具現化した“OPEN 2200”の第1号プロダクトとして,2200/500シリーズが続く.このような情勢のもとで,OS 1100も大きく変貌する.以下の章で,開発・実行環境,日本語処理,オープン化について記述する.

3. 開発・実行環境

3.1 プログラミング言語

各種のプログラミング言語への対応はオペレーティング・システムの主要な機能の一つである.その構成要素はコンパイラと実行時ルーチンのライブラリである.

CODASYL(the Conference on Data Systems Language;データシステムズ言語協議会)は,プログラミング言語COBOLに関する著作物の冒頭にCOBOLの開発者に対する謝辞を掲載することを要求しているが,その中でCOBOLの原仕様書の作成に当たって利用することの許された著作物の一つとして“FLOW-MATIC”を掲げている[*1].

このように,スペリーランド社はプログラミング言語の標準化の礎を築いた一員であった.シリーズ2200のプログラミング言語の提供において,標準への準拠はその基本方針の一つである.

1107および1108システムに用意されたコンパイラは,CODASYL COBOL(EXEC 8 COBOL),ANSI COBOL(Fieldata COBOL),Fortran IV,Fortran Vであったが,いずれも当時の標準に準拠していた.とくにFortranは非常に優秀なコンパイラとして評価され,多くの科学技術計算に利用された.

1970年代の前半,1110システムや1100/20,/40,/10システムが提供された時代に,次世代のコンパイラ群が登場した.この世代のコンパイラは7・8ビットの文字コードを前提としていたので,(COBOLやFortranの場合,前世代のコンパイラと区別するため,)それぞれ“ASCII”を冠して,ASCII COBOL,ASCII Fortranと名付けられた.この他に新しいプログラミング言語として,PL/1およびPLUSが提供された.COBOL,Fortran,およびPL/1は,当時の最新の米国標準規格である,ANSI X.3.23

-1968 COBOL, ANSI X. 3.9-1966 Fortran, ANSI X. 3.53-1976 PL/1 に準拠していた. したがって, これら米国標準規格に対応する国際規格(International Standard)および JIS にも準拠していた. PLUS は, Programming Language for Univac System の頭辞語 (acronym) で, 当時のスペリー社の二つのコンピュータ・アーキテクチャであるシリーズ 1100 と中小型機の 90 シリーズに共通のシステム記述用として開発された高級言語である. もっぱら, システム・ソフトウェアを記述するために開発され, 一般ユーザには提供されていない. この他に, 提供されたコンパイラとしては, BASIC, APL, アセンブラ, メタ・アセンブラがある.

ASCII COBOL は画期的なコンパイラで, シリーズ 1100 のビジネス・アプリケーションへの利用の道を開いた. 機能としては報告書機能等, ANSI, ISO, および JIS の最高水準を提供した. 1110 システム, 1100/60 (Vanguard) システム等にバイト命令が導入されると, これらを使用する実行コードを生成するように改良され, 36 ビット 1 ワードであるシリーズ 2200 では不利とみなされていた COBOL コンパイラを十分競合力のあるコンパイラに発展させた.

前世代の COBOL および Fortran, さらに ASCII COBOL は外部のソフトウェア開発会社にその開発を委託していたが, ASCII Fortran 以降は自社製のコンパイラである. 新しいハードウェア, オペレーティング・システムからの要求, 新しい標準規格の制定, 新しく実用化されたコンパイル技法の確立等の状況で, コンパイラを自社で開発する決断がなされた. 開発作業は既存のコンパイラの大幅な改良であり, 新しい言語, PL/1 や PLUS のコンパイラを新たに開発する必要もあった. これらのコンパイラが互換性のある体系となるように, 全てのコンパイラを一つのプロジェクト内で開発することになった. 異なる言語のコンパイラ間での共通性や互換性を強調した開発思想のもとで, 一連のコンパイラが並行して開発された. この開発思想を「共通コンパイラ・モデル」と呼ぶ. 共通コンパイラ・モデルに従って開発されたコンパイラは, PLUS, ASCII Fortran, および PL/1 である.

まず, PLUS 言語とそのコンパイラおよび実行時ライブラリが開発された. プログラムの可搬性のために PLUS では PLUS ソース・コードの中にアセンブラのソース・コードを記述することが許されていない. (一般のシステム記述用の高級言語では, しばしば, 一つのシンボリック・エレメントに高級言語のソース・コードとアセンブラのソース・コードが混在する状況を回避できなかった.) そのため PLUS では, 混在するアセンブラ・コードでレジスタや記憶領域の内容が変更される可能性を考慮する必要がないので, 最適化を十分に行うことができる. また, 高級言語で記述されたソフトウェアは, アセンブラで記述されたソフトウェアに比較して, コード・レビューが容易, すなわち不具合の発見・修正が容易であり, したがって, 開発・改造期間の短縮やリリース初期からの安定性を期待できる.

PLUS の開発に引き続いて, ASCII Fortran のコンパイラの開発が進められた. 同時に, PLUS の改良も進められ, ASCII Fortran の開発も, 新しい PLUS に切り替えられた. 若干遅れて, PL/1 の開発も開始された.

「共通コンパイラ・モデル」の開発思想に基づき, これらのコンパイラは全て四つのフェーズ, 構文分析 (第 1 フェーズ), 文脈分析 (第 2 フェーズ), 最適化 (第 3 フェ

ーズ)，およびコード生成（第4フェーズ)，から成る．構文解析は言語依存であり，ほぼ独立している．第2フェーズから第4フェーズまでは出来るだけ共通な設計が採用されている．各言語のコンパイラ開発の順番も，言語仕様の比較的単純なものからより複雑なものへと，行われた．この結果，各コンパイラと実行時ライブラリは，物理的には独立であるが，共通部分の多いものとなった．「共通コンパイラ・モデル」は，次世代のコンパイラ，UCSへ発展する．

シリーズ2200の第三世代のコンパイラは，1982年に発表された超大型機1100/90システムの拡張アーキテクチャに対応するコンパイラ群である．コンパイラ・システム全体としてUCS（Universal Compiling System，汎用コンパイル系）の名称が与えられた．プログラミング言語として，従来のCOBOL，Fortran，およびPLUSに加え，新たにPascalおよびCが開発された．前世代と区別するために“UCS”を冠して，たとえば，UCS COBOL，あるいは省略して，UCOBと呼ぶ．UCSの特徴の第1は，コンパイラの一部と実行時ライブラリを言語間で共通化したことである．UCSは前世代の「共通コンパイラ・モデル」を発展・完成させた．すなわち，「共通コンパイラ・モデル」では，ASCII FortranとPL/1では，コンパイラと実行時ライブラリは共通の設計であったが，物理的には，共通要素はもっていたが別々であった．UCSでは最適化（第3フェーズ)，コード生成（第4フェーズ)，および実行時ライブラリが物理的にも共通になった．システムには，唯一つのコンパイラと実行時ライブラリとがあればよい．必要なプログラミング言語によってフロントエンドと呼ばれる言語依存の部分のみを導入する．第2に，言語としてCOBOLが加わった．前世代のASCII COBOLは，アセンブラで開発された他社製品であったが，UCSにおいて，高級言語で自社開発されたCOBOLコンパイラが初めて提供されることになった．ユーザ・アプリケーション・プログラムの大半を占めるCOBOLのUCS化は，開発・改造期間の短縮や安定性の向上に役立つことが期待された．その後，C言語の開発にもUCSの開発方法が適用され，UCS CがUCSの一員となった．UCSの概念図を図10に示す．

UCSの開発者にとっての利点は明らかである．新しい言語のコンパイラ開発が，一般的に短期間で安価に実現できる点である．バック・エンドが共通化されているので，複数の言語のコンパイラを開発する場合，開発費の総和はコンパイラを個々に開発する場合の開発費の総和よりも小さくなる．いったんUCSが開発されると，新しい言語のコンパイラの開発費も小さくなる．すでに存在するバック・エンドを利用できるからである．同様に保守費用も小さくなる．

UCSのユーザにとって，汎用性は不利にはならず，むしろ利点となる．すなわち，

- ●実行時システムが共通で，言語間の呼出しのインタフェースも統一されているので，アプリケーションを複数の言語で開発することができる．アプリケーションのある部分にその部分の処理に最適なプログラミング言語を選択できる．
- ●各言語のコンパイラの技術を集大成したコンパイラによって効率のよいプログラムが開発できる．
- ●バック・エンドの改良，たとえば最適化の改善等をどの言語でも享受できる．
- ●共通のデバッグ・ツールが使用できる．

以上の三世代のプログラミング言語コンパイラの特徴として，標準への準拠を挙げ

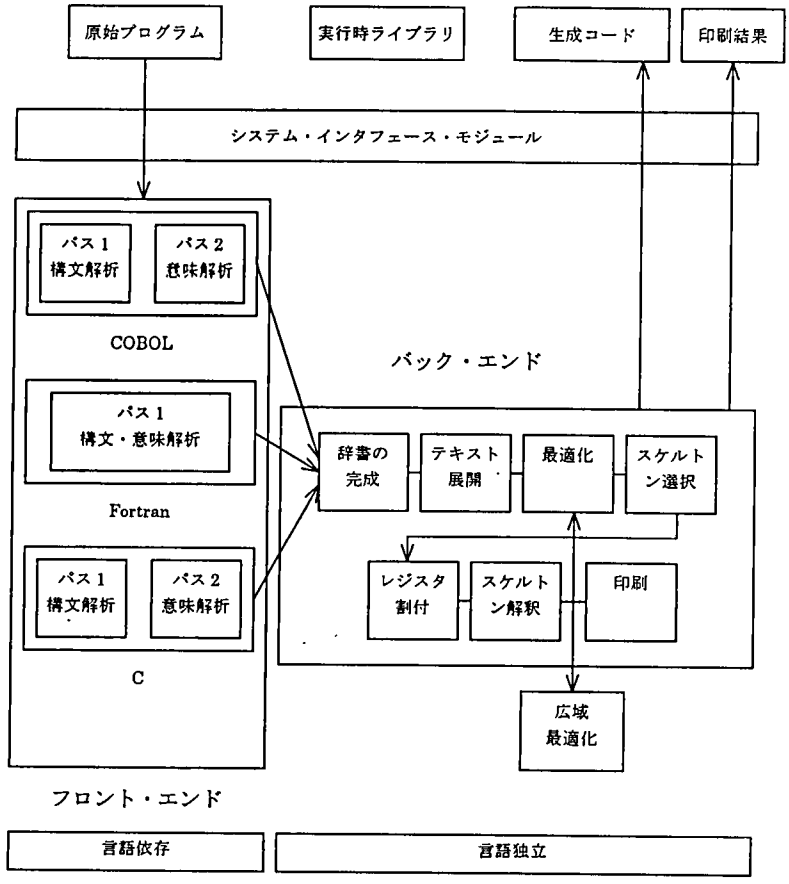


図 10　UCS の概念図

た．標準は時代と共に変るものである．プログラミング言語の標準もこの 34 年間に何度か改訂された．COBOL および Fortran には三つの版があり，版によって言語仕様が異なる．言語仕様の変更はそれ自体，進歩・改良であって望ましいことであるが，言語仕様の非互換が問題となる．ユーザにとって，言語仕様の変更が拡張のみであれば問題はない．しかし，実際には，前の版とは互換性のない言語要素が存在し，前の版に準拠するユーザ・プログラムを，新しい版の標準に準拠させるには，ユーザ・プログラムの書換えが必要になる．シリーズ 2200 のコンパイラはこの問題に一世代のコンパイラが，二つの標準の版をサポートすることで対処してきた．すなわち，ASCII COBOL は，ANSI の 1968 年版と 1974 年版とをサポートする．また，UCS COBOL は，1974 年版と 1985 年版とをサポートする．二世代のコンパイラが長期間にわたって同時にサポートされるので，ユーザはその間に新しい版の言語標準に対応するようにプログラムを変更することが可能になる．

3.2　データベース・マネージメント・システム

1971 年終り，データベース管理のプロダクト DMS 1100（Data Management System　1100）がリリースされ，シリーズ 1100 はデータ管理の新時代に突入した．DMS 1100 は CODASYL データベース作業班（CODASYL Data Base Task Group；

DBTG）の仕様に準拠していた．データベースの特性（スキーマ）と保護を定義するデータ記述言語（DDL）と，エンド・ユーザから使用できる論理的なサブセット（サブスキーマ）を記述する言語(SDDL)とを備え，データ操作用言語(DML)を，COBOL, Fortran, PL/1の言語仕様の拡張として備えていた．各DMLは機能は同一であるが，構文がそれぞれの言語（親言語）に適するようなスタイルに変更されている．

データベースの記憶構造には，ハッシュ(CALC)，直接編成，索引付き編成，ポインタ・チェイン編成，ポインタ・アレイ編成，索引付きポインタ・アレイ編成の6種類がある．データ管理ルーチンはマルチスレッドで，バッチ，デマンド，トランザクション形態のプログラムでデータベースを共有し，同時にアクセスできる．アクセスの衝突はロックメカニズムによって制御される．

TIP 1100（トランザクション・インタフェース・パッケージ）は，ユーザのトランザクション・プログラムがリアルタイムで動くための，一般化されたフレームワークを提供する．TIP 1100はEXEC 8の通常の機能に，トランザクション・プログラム・サービス（たとえば，プログラムのロード，初期化，終了等）を提供する．TIP 1100はDMS 1100データベースを処理することができる．さらに，TIP 1100は特殊なファイル制御機能をもち，膨大なデータを含むファイルを高速処理することができる．これは，航空機の座席予約システム等に使用される．

CODASYLデータベース作業班がCODASYL方式（ネットワーク方式）のデータベースの仕様を発表した1970年代初めに，データの関係モデルが提唱された．関係モデルの目的は，データの独立性を達成し，ユーザに簡単なデータの見方を提供し，データ管理者の負担を軽くすること，とされる．1970年代にはいくつかの実験的なシステムが登場した．IBMのSystem Rはその集大成といわれ，PL/1等の親言語から呼び出すSQLの基本的なアイデアも含まれていた．

シリーズ1100の関係モデルのデータベース・システムは，RDMS 1100 (Relational Database Management System 1100)として，リリースされた．ここにシリーズ1100では2種類のデータ・モデルのデータベース管理システムが提供されることになった．このような場合，一つのユーザ・アプリケーションが複数のデータ・モデルを使用する場合が出てくる．このとき，異なるデータベース管理システム間のロックやデータベースの回復を同期をとって実行する機能が要請される．これに応えるのがUDS (Universal Data System)である．UDSは，CODASYL型（ネットワーク型），関係型，および階層型モデル(SFS 1100)をサポートする統合データ管理システムである．統合化の結果，メッセージやデータベースの回復が一元化され，ユーザによるシステム構築が容易に，データベースの管理・運用が確実・効率的になった．また，UDSの統合リポジトリ機能によって，運用経費の削減にも寄与した．UDSの概念図を図11に示す．

UDS 1100コントロールは，UDSの中核としてすべてのデータ・モデルについて，スレッド（処理プログラム）制御，ロックやキューの管理，データベースの入出力管理，記憶域管理，データディレクトリ管理，およびデータベースの回復を統合的に行う．IRU 1100 (Integrated Recovery Utility 1100)は，障害時にデータベースやトランザクション・メッセージを回復し，データベースの整合性を保障するために使用す

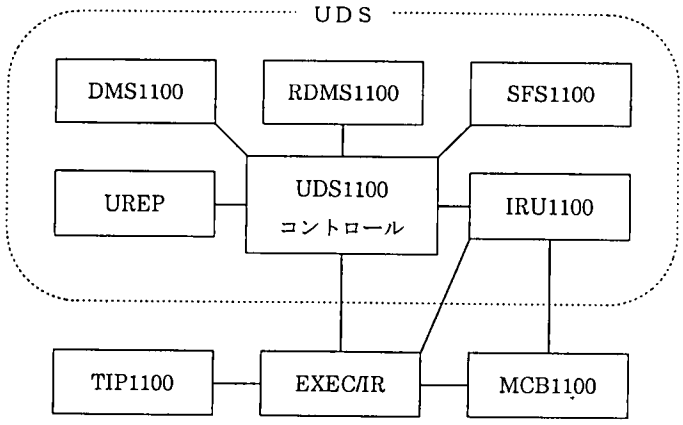


図 11　UDS の構成

る統合回復ユティリティである．IRU 1100 は，このほかに，データベースの回復に備え，データベースを定期的にディスクやテープに保存したり，ファイルの状況を検査する機能などを持つ．IRU 1100 はステップ制御，オーディット制御，MCB(Message Control Bank) および UDS コントロールのデータベース回復機能と共同して回復処理を行う．

UREP (Unisys Repository) は，実体関係モデル (Entity Relation Model; ER Model) によって管理対象をモデル化し，UREP 言語でリポジトリに投入された情報にそって管理する．UREP はユーザの資源の有効活用を支援するツールである．UREP は UDS のシステム辞書の管理 (アプリケーション・グループの構成，ファイルの記憶域定義，リレーショナル・テーブル定義等) に使用される．

3.3　ソフトウェアの統合化と EASE

1979 年の 1100/60 (Vanguard) システム導入の際には，オペレーティング・システムを，言語プログラム，データベース管理プログラム，データ・コミュニケーション・プログラム，1100 サポート・プログラム，およびシステム・サービス・プログラム，の五つに分類した．ここには分類はあるが，互いの関連や全体としての目的意識は希薄である．しかし，OA や分散処理のようなコンピュータ利用環境では，アプリケーションの開発・保守の生産性の向上が急務となり，コンピュータ資源をより容易に，有効に，安全に利用する開発・実行環境が要請されるようになった．

1982 年の 1100/90 導入を期して，コンパイラやデータベースのようなシステム・ソフトウェアが統合化され，EASE (Eleven Advanced Software Environment) と称して提供された．オペレーティング・システムの統合化・体系化は，それを提供する側にとっても，機能の充実，機能の重複の防止，保守性の容易さ等の利点がある．新しいソフトウェア体系 EASE は，四つのソフトウェア群を統合化したものであった．

1) ユーザとの新しい対話型インタフェースとして IPF 1100 (Interactive Programming Facility) を提供

2) エンド・ユーザ・コンピューティングのツールとして，MAPPER 1100 に加えて，QLP 1100 を提供

3)　プログラム開発・実行環境としてUCSを提供

4)　データを統合的に管理するシステムとしてUDSを提供

3.4　第4世代言語とCASE

EASEはアプリケーションの開発・保守環境を統合化し，生産性の向上を実現した．また，エンド・ユーザ・コンピューティング・ツールによってユーザによるアプリケーション開発が一部可能となった．しかし，大規模アプリケーションの開発は依然として専門家の役割であった．必要なアプリケーションの数は飛躍的に増加し，コンピュータの処理能力に余裕があるにもかかわらず，開発者，たとえばプログラマの数が不十分で開発できないアプリケーションのバックログが増加していった．また，利用者と開発者の乖離，すなわち利用者が望むようなアプリケーションができないことも問題であった．さらに世の中の変化のテンポが早まり，開発中あるいは開発後のアプリケーションの仕様の変更，機能の追加や変更，が頻繁に発生するようになった．開発・保守作業の飛躍的向上がなければ，これらの問題に対処できない．

このような状況のもとで，生産性の向上およびエンド・ユーザの開発参加を目的とした第4世代言語（Fourth Generation Language；4GL）が登場した．4GLはアプリケーション開発過程のうち，プログラミング以降に適用される．要求仕様の作成，システム設計のようなプログラミングの前段階を対象としたツールも提供されはじめ，統合的なCASE（Computer Aided Software Environment）が現実的になってきた．

シリーズ1100では，MAPPER 1100およびQLP 1100が提供されてきたが，大規模トランザクション・システムの構築用の4GLとして，LINCがリリースされた．LINCはAシリーズからシリーズ1100に移植されたのであるが，シリーズ1100の中核の技術（UCS，UDS，TIP等）を活用しており，COBOL等の3GL（Third Generation Language）によって開発されたトランザクション・システムと比較しても機能や性能に遜色のないアプリケーションが構築できる．

4.　日本語処理

4.1　文字コード

シリーズ2200のオペレーティング・システムは当初（1960年代），独自の6ビットFD(Fieldata)コードを前提としていた．FDコード64個の文字種は英大文字，数字，および記号類であった．英小文字や片仮名は含まれていない．片仮名はシフト・コードを使って表現され，片仮名を印刷するために特殊なルーチンが使用された．

標準の世界では情報交換のための7・8ビットのコードの制定が進んでいた．米国では1962年にASCII（American Standard Code for Information Interchange）が制定され，国際的には1967年にASCIIコードに基づき，各国の文字に配慮したISO R 646が制定され，日本では1969年にISO R 646に準拠し，片仮名を追加したJIS C 6220情報交換用符号を制定した（後にJISに情報処理部門が新設されると内容を変えずにJIS X 0201となった）．一方で，当時はIBM 360系のEBCDIC(Extended Binary Coded Decimal Interchange Code）が事実上の標準となっていた．

シリーズ2200の文字コードを独自のFDコードから標準規格のASCIIコードに変

更する決定がなされた．ASCII コードベースで開発された最初の製品として，1973 年に ASCII COBOL がリリースされた．日本では，JIS X 0201 の 8 ビット・コードの処理が可能となり，ユーザにとっては，片仮名の入力，画面への表示，紙への印刷が，特別な考慮なしに行えるようになった．(ASCII COBOL は，FD コードも処理できるように作られていた．)プログラミング言語やデータベース管理システムの標準規格準拠に続いて，文字コードも標準規格に準拠することになった．この決定は，シリーズ 2200 のオペレーティング・システムが標準に準拠していくという方針を明確に示すと共に，後の日本語処理，国際化，オープン化を容易にする，重要な決定であった．ただし，漢字を加えた完全な日本語処理にはなお 6 年を要した．

4.2 日本語処理システム LETS-J

プリンタやディスプレイの進歩によって漢字の印刷や表示が可能になってくるとコンピュータによる日本語処理システムの開発が始まった．当社では，1974 年に業界初のコンピュータによる漢字情報処理システムをリリースした．その後も漢字処理の超高速/高速プリンタ，漢字ディスプレイ，漢字ワークステーション等のハードウェア，日本語の入力・編集・ソート・印刷等の日本語処理ソフトウェアの拡充を行ってきた．

1976 年には，漢字コードの標準規格 JIS C 6220（後の JIS X 0208）が発行された．これは，7•8 ビットの JIS X 0201 を 2 バイトに拡張した構造を持っていた．各社から汎用機による日本語処理システムが発表された．

当社は，先進的なオフライン日本語処理の技術をメインフレームによるオンラインの日本語処理に注ぎ込み，1980 年に，シリーズ 1100 等による日本語処理システムがリリースされた．これは，LETS-J（Linguistically Extended Technology on Univac System Japanese＝Let us Japanese）と命名された．ハードウェアとして 0780 型日本語プリンタおよび UTS 5000 型漢字ディスプレイ・ターミナルを備え，日本語処理特有の処理のためのソフトウェアを新たに開発すると共に，オペーレーティング・システム全体に日本語処理のための改良を加え，従来のデータ処理と同一の水準で日本語を含むデータの処理環境を実現した．文字コードとして，JIS X 0201 を採用していたため，漢字コード JIS X 0208 が無理なく実装でき，当社拡張文字やユーザ拡張文字の領域も十分に取れ，後年，補助漢字 JIS X 0212 が制定された際も，早期の実装が可能となった．日本語対応された主要なソフトウェアには，日本語 COBOL，日本語 Fortran，日本語 PL/1(以上プリプロセッサ)，日本語 DMS，日本語 QLP，日本語 CTS 等がある．

これら「日本語」を冠するソフトウェアは，もともと日本語処理機能を持っていない既存のソフトウェアに日本語機能を付加するための，独立で補完的なソフトウェアか(プリプロセッサなど)，日本語処理機能を埋めこんだソフトウェアである．先に述べたような EASE のもとに統合されたソフトウェア，UCS や UDS 等は，初めから日本語処理機能を提供するように開発された．

5. オープン化

5.1 シリーズ 1100 上の UNIX

UNIX をシリーズ 1100 上で稼働させる努力は以前から進められおり，一部のユー

ザで使用されていた．1986年になって一般ユーザにもリリースされた．このUNIXはAT&TのSystem V Release 3に基づいてシリーズ1100に実装したもので，SX 1100と命名された．しかし，日本においては，日本語処理やアドレス空間の制約等があり，一部の教育機関，研究所での実験的な使用にとどまった．

SX 1100の機能を利用して，NFS 2200（Network File System 2200）が提供された．この製品は業務に使用されている．

5.2 UAと2200/900

1987年，IBM社がSAA（System Application Architecture）を発表すると，他社もこれに追随した．これら1980年代後半に発表されたアーキテクチャはユーザを自社製品で囲い込む意図を持った独自のアーキテクチャであった．

当社は1990年に，UA（Unisys Architecture）を発表したが，オープンに対応したアーキテクチャであった．

オープンとは何か．X/Openによれば，オープン・システムとは，「特定のベンダーに依存せず，すべての人々が共通に利用可能な標準に基づくシステムおよびソフトウェア環境」である．ガートナ・グループによれば，オープン・システムとは，「多様性を持ち，広い範囲で受け入れられる業界標準に準拠した相互運用性を持ち，異機種プラットフォームで使えるポータブルなシステム」である．「オープン・システムとは，ユーザの仕事に必要な情報を常時アクセスできるようなシステム」との多少毛色の変わった定義もある．

当社はオープン・システムの戦略として，確立された業界標準や新しい業界標準に基づいて既存のシステムをマルチ・ベンダー・システムへと随時統合していく戦略「情報システム・アーキテクチャ」を推奨している．

UAは企業の「情報システム・アーキテクチャ」を作成するためのガイドラインとして位置付けられる．すなわち，「UAとは情報技術を企業のビジネス戦略に適用するためのガイドラインで，UAによって企業の情報システムの設計図となる情報システム・アーキテクチャが作成できる．」

UAの特徴は，マルチ・プラットフォームの統合を含む「包括性」，オープンな標準（ISO，POSIX等），業界標準（Windows，NetWare等）とベンダー固有の標準（DCA等）を取り込んだ「標準ベース」，および技術をビルディング・ブロックとして定義する「モジュール性」である．

UAの構成要素は，三つの「プラットフォーム」，三つの「機能クラス」，および五つの「サービス」である．

三つの「プラットフォーム」とは，インフォメーション・ハブ（情報中枢），ネットワーク/地域サーバ，およびワークステーション/ワークステーション・グループである．インフォメーション・ハブの役割は，基幹データベース管理，基幹リポジトリ管理，高信頼性アプリケーション処理，バッチ処理，基幹トランザクション処理，定型処理等である．これらの役割を果たすため，インフォメーション・ハブは，その特性として，無停止連続運転，高度なセキュリティ，限りない成長性，統合的システム管理，大規模トランザクション処理，大規模データベース管理をもつ．

三つの「機能クラス」は，機能を標準（ISO，X/Open等）の「オープン・クラス」，

業界標準（事実上の標準）（Windows，NetWare等）の「共存クラス」，および当社固有（4GL，OLTP等）の「プレミアム・クラス」である．共存クラスとプレミアム・クラスとは，次第にオープン・クラスになっていくと考えられる．あるいは，共存クラスとプレミアム・クラスは将来のオープン・クラスの候補である．機能クラスがオープンになるに従って，ユーザのシステムが徐々にオープン・システムになっていく．

五つの「サービス」は，「アプリケーション・サービス」，「インフォメーション管理サービス」，「分散システム・サービス」，「システム間接続サービス」，および，「システム管理サービス」である．これらのサービスを具体化する三つのフレームワークを発表した．その三つとはASDF（Advanced Solution Development Framework, 1992年），ACCF（Advanced Cooperative Computing Framework, 1993年），および，ASMF（Advanced System Management Framework, 1994年）である．

1991年に発表された超大型機2200/900シリーズは，UAのインフォメーション・ハブとしての役割を担うべきシステムとして開発されたもので，そのために拡張処理アーキテクチャ2200/XPA（2200 eXtended Processing Architecture）が新たに導入された．2200/XPAは，三つのサブ・アーキテクチャ，拡張中央処理アーキテクチャXCPA（eXtended Central Processing Architecture），（2200/600シリーズですでに導入済みの）拡張トランザクション処理アーキテクチャXTPA（eXtended Transaction Processing Architecture），および拡張入出力処理アーキテクチャXIOA（eXtended Input/Output Processing Architecture）である．

拡張中央処理アーキテクチャXCPAは，拡張マルチプロセッサ・システム（最大8台），サイバー空間（システム空間），大容量記憶装置（最大2ギガ・バイト）等によって超大規模・超高速処理を実現する．サイバー空間とは，システム内の全てのユーザ・プログラムのアドレス空間（各ユーザごとに最大175ギガ・バイト（0.5ギガ語））と主記憶装置の物理アドレスとを媒介する巨大な仮想空間として導入され，その大きさは最大560テラバイト（560×1012バイト，140テラ語）である．概念図を図12に示す．

サイバー空間を導入した目的は，

- 新しいページング方式を，従来のバンク方式と調和する形で導入すること，
- 主記憶容量の巨大化と既存のユーザ・アプリケーションおよびコンパイラ等のシステム・プログラムの互換性（新アドレス空間でも稼働すること）とを両立させること，
- ファイルをサイバー空間に割り当てる（サイバー空間をファイルのように利用する）機能（メモリ・ファイル機能）によって，入出力処理を高速化すること，

である．サイバー空間を活用してバッチ処理効率を向上させる機能を次に示す「サイバー・バッチ・システム」として提供している．

- EXFILE：ファイルをサイバー空間に展開し入出力処理を高速化する．
- EXPIPE：バッチの連続ラン間のデータの受渡しをサイバー空間を介して行い（パイプ），連続ランの並行動作により，処理時間を短縮する．
- ESORT：サイバー空間をソート作業領域として活用し，ソートを高速化する．

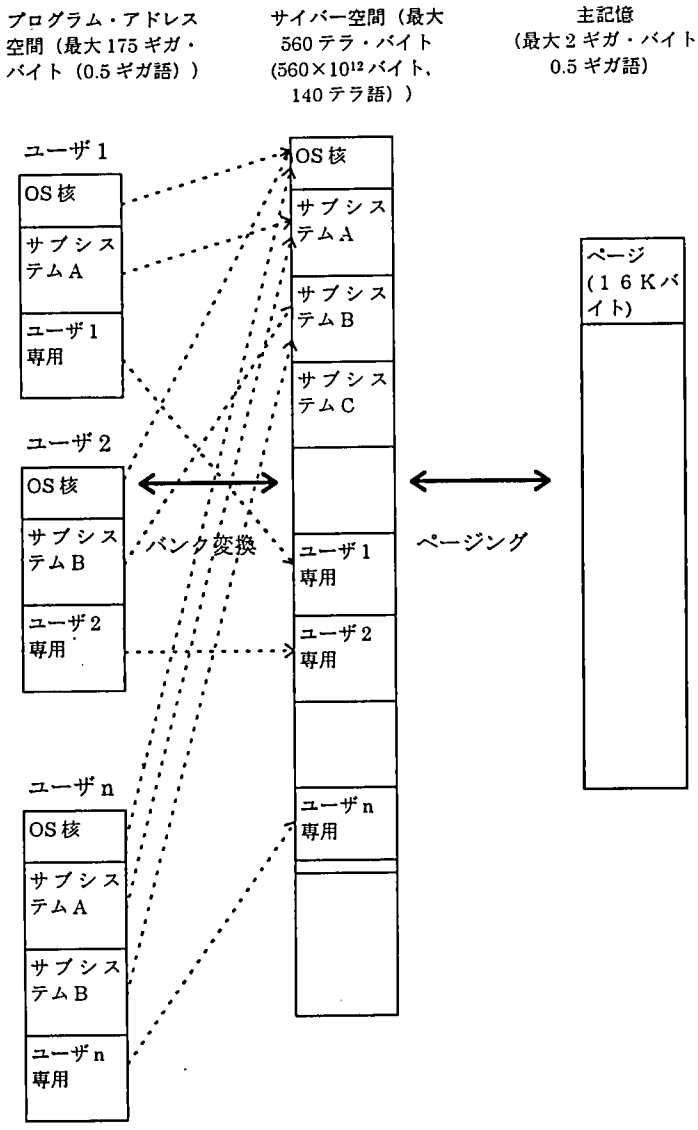


図 12　サイバー空間

5.3 OPEN 2200

シリーズ2200は，UAに沿ってオープン対応を進めてきたが，オープン対応を強化すべく，OPEN 2200を発表した（1993年）．

OPEN 2200は，オープンシステムと固有システムとの共存・協調をシリーズ2200上に具現化したもので，メインフレームの特徴である，無停止連続運転，高度なセキュリティ，限りない成長性，統合的システム管理，大規模トランザクション処理，大規模データベース管理を強化するものである．さらにオープン・サービス機能，すなわち，オープン・デベロップメント，オープン分散アプリケーション，オープン・データ・マネージメント，オープン・ネットワーキング，オープン・マネージメント等に対応するものである．

OPEN 2200 の一番手は，2200/500 シリーズである．ハードウェアとしては，2200/400 で採用した CMOS VLSI 技術を継承し，2200/900 シリーズの 2200 XPA（拡張処理アーキテクチャ）を装備した中大型機である．

1996 年には，超大型メインフレームとしては初めて，CMOS VLSI を採用した ITASCA 3800 シリーズが発表された．ITASCA でのオープン対応では，

- オープン・データ・アクセス：PC からメインフレームのデータベースをアクセスする．
- Unix 環境の提供：メインフレームに Unix のアプリケーションを稼働
- クライアント/サーバ環境の提供：分散トランザクション，分散ファイル，分散プリント等，

が拡充される．

6. おわりに

シリーズ 1100・2200 と OS 1100 の 34 年間の歴史を足早にたどってみた．この間に，シリーズ 1100・2200 と OS 1100 は，ハードウェアもソフトウェアも著しい機能・性能の向上を達成してきた．シリーズ 1100 を推し進めてきた要素は，ハードウェア技術の発展，ソフトウェア技術の発展，安価な PC やワークステーションの出現，ネットワークの発展，情報システムの高度化等が考えられる．

シリーズ 1100・2200 は，大きなアーキテクチャの変更を 2 度行った．すなわち，1982 年の 1100/90 システムでの「拡張アーキテクチャ」および 1991 年の 2200/900 シリーズでの「拡張処理アーキテクチャ」2200/XPA である．アドレス空間が飛躍的に拡張され，処理効率が向上した．シリーズ 1100 の特徴は，旧アーキテクチャも維持したことである．いずれの拡張においても，既存のアーキテクチャが継承され，ふるいアーキテクチャでのユーザの資産（プログラムおよびデータ）が保護された．

シリーズ 1100・2200 は，歴史的に大きな技術変革があったにもかかわらず，唯一つのオペレーティング・システムを維持してきた．また，中型機から超大型機までの幅広い拡張性にもかかわらず，単一シンボリックの方針を堅持してきたことも特徴として挙げられる．

OS 1100 は，ユーザ・プログラムおよびコンパイラやデータベース管理システムのようなシステム・ソフトウェアをコンピュータの物理的性格から「絶縁」しているので，ハードウェアの発展がソフトウェアの変更なしで享受できる．ソフトウェア技術の発展，ネットワークの発展，安価な PC やワークステーションの出現，情報システムの高度化等に関連して，OS 1100 の特徴として統合化とオープン化が挙げられる．統合については，最初は，コンパイラの統合（UCS）やデータ管理システムの統合（UDS），次にこれらを統合した開発・実行環境（EASE）が提供された．さらに UA では，OS 1100 の種々の機能が，ASDF 等のフレームワークを構成するビルディング・ブロックとして統合されている．オープン化も OS 1100 の基本方針であった．プログラミング言語や文字コードの標準規格への準拠は，オープン対応の第一歩であった．UA では，機能全般についてオープン化が明確に定義され，それに準拠した製品の具体化が進められている．

OS 1100の将来は，これまでの延長線上に描くことができる．ユーザ資産の保護しつつ先進的な技術を提供すること．そのために，OS 1100は堅持されるべきであるし，UAおよび各フレームワークは，標準や技術レベルを見通して改定され，OS 1100の製品や機能に具体されるべきである．ただし，進展はこれまでよりも広範にわたり，急激になることが予想される．

＊1　FLOW-MATIC（スペリーランド社の商標），Programming for the Univac I and II (R), Data Automation Systems, スペリーランド社1958年，1959年，著作権．」

執筆者紹介　伊　藤　龍　彦（Tatsuhiko Ito）

昭和47年早稲田大学大学院理工学研究所数学科卒業．同年日本ユニシス(株)入社．以来シリーズ2200の言語プロセッサ，日本語処理システムなどの開発，受入れおよび保守に従事．現在，システムプロダクト部オープンエンタープライズサーバ開発推進室に所属．

Aシリーズオペレーティング・システムの発展

The Evolution of A Series Operating System

鈴 木 秀 明

要 約 ユニシスAシリーズのオペレーティングシステムであるMCPは，システム資源をいかに無限にあるように扱うかという「資源の仮想化の実現」を基本に据えて開発されてきた．これは，プログラムによるコンピュータ資源の利用において，その扱いの抽象度を高め，論理的に扱えることを狙ったものであり，オペレーティングシステムの基本技術を形成する．一方，プロセッサや入出力機器の高速化とメモリの巨大化の急速な進展のなか，MCPの誕生当時とは桁違いの性能・規模の資源を扱う，新たな技術の開発と実装を行い，大容量データベース・大規模トランザクション処理を頑強なシステムのもとに実現してきた．そして今日，オープン環境での異種システムとの接続と，「異なるオペレーティングシステムとの統合」技術を実装する段階に入った．

本稿では，MCPの発表当初から現在に至る，MCPおよびAシリーズで実現してきた技術の変遷を述べるとともに，今後のオープン・エンタープライズ・サーバを支える技術を展望する．

Abstract MCP, the operating system of Unisys A Series, mainframes, has been developed with the concept of "virtual resources" whereby system resources are considered as unlimited. Aiming for a high-level of abstraction and more logical resource utilization whenever a program uses computer resources, this concept forms the basic technology of the operating system. Given the rapidly increasing speed of processors and I/O devices and the rapidly increasing capacity of memory available, new technologies have been developed and implemented, enabling MCP to handle resources with far more exceeding level of performance and system scale than those when MCP was first introduced. Large-scale database and high-volume transaction processing were thus realized on the robust systems that run MCP. Now we have reached the stage where we will implement the technology for integrating with different operating systems in order to facilitate connections with heterogeneous systems in the open computing environment.

This paper describes the transition of the technologies used in the MCP and A Series mainframes from the start of MCP's development till the present, and gives a view of the technology that will support open enterprise servers of the future.

1. はじめに

ユニシス（当時のバロース）は，Aシリーズの最初の機種であるB 5000を設計するにあたり，E. I. オーガニックが「計算機システムの構造」の中で「バロースの設計者達は，オペレーティングシステムに対するアルゴリズムが，そもそもブロック構造をもっているものとみなし，その構造を反映できる言語（Algol 60の拡張版）を用いて，これらのアルゴリズムを記述することにした．」（「計算機システムの構造」バロース大型計算機シリーズ，E. I. オーガニック著，土居範久訳，共立出版）[1]と記述しているように，このブロック構造を無理なく実現するためにハードウェアにスタック機構を取

り入れ，さらに，その上で稼働するオペレーティングシステム（OS）であるマスタコントロールプログラム（MCP）の開発では，アセンブラ言語を使用するのではなく，ブロック構造を自然に表現できるALGOL 60を基本とした高水準言語を用いた．このB 5000とMCPは1961年に発表された．このシステムの機構上の本質的なものはRobert S. Bartonによるものである．その後，B 5500，B 6500ハードウェアの開発とともに，多重プログラミング，多重プロセッシング，仮想メモリの実現がなされ，現在のMCPの原型が形成された．

1960年代におけるMCPの開発は，業務プログラムの実行に対し，制御の対象としているシステム資源をいかに無限にあるように扱えるか，すなわち，「資源の仮想化」の技術の開発に注力されていた．プロセッサに関しては，プログラムの実行に必要な変数などの格納領域としてのスタックを論理プロセッサとして用い，論理プロセッサの物理プロセッサへのマッピングとして多重プログラミングを実現した．また，メモリ制御にセグメンテーション方式に基づく仮想メモリを実現し，二次記憶媒体を実メモリの延長として扱えるようにした．入出力装置の扱いにおいて，カード読み取り装置やプリンタなどの低速装置を，ディスクやテープなどの高速装置に一時的に代替させる「疑似装置化」も，入出力装置の仮想化の技術の一つである．

1970年のB 6700/B 7700の発表にともない，この仮想化の技術は，配列や構造体型のデータにも適用された．この技術をMCPに代表される制御系のソフトウェアが利用することによって，規模に応じたデータ量を動的に（実行時に）扱うことが可能となった．たとえば，MCPの扱うハードウェア機器構成の大小，同時に扱うファイル数やトランザクション数などは，その時点の規模に応じて最適に扱え，制御系ソフトウェアのプログラミングの可用性を格段に高めることとなった．

OSの仮想化の技術は，OSの善し悪しを決める基本技術であり，今日でもその適用範囲の拡大がなされている．本稿では，MCPの誕生期における「仮想化の技術」を出発点に据え，処理性能の飛躍的な向上を果たしながら，その技術がどのように変遷してきたかを述べる．最後に，頑強で，大容量データベース・大規模トランザクション処理システムやバッチ処理システムを扱う技術に加え，今日求められているオープンサーバとしての期待に，それらの技術がどのように生かされているかを述べる．

以下に1961年にMCPが発表されてから現在に至る，主な機種とその時々の新機能・技術を年代順に記述する．

1961年　B 5000 MCP（Master Control Program）発表

1964年　B 5500発表

1. 多重プログラミングの提供
2. 多重プロセッサ（マスタ/スレーブ方式）の提供
3. 20ビットアドレスによる6メガバイト実メモリ空間の提供
4. セグメンテーション方式による仮想メモリ
5. 主記憶保護（タグ，不当索引誤り等のハードウェア検出）
6. TSSの提供

1966年　B 6500発表

1970年　B 6700/B 7700発表

1. 平等プロセッサ方式による多重プロセッサの提供

2. タスキングの提供
3. 入出力専用プロセッサの提供
4. データ通信専用プロセッサの提供
5. 高水準ジョブ制御言語（Work Flow Language）の提供
6. ネットワーク型データベースの提供

1976年 B 6800/B 7800 の 800 シリーズ発表
1. 密結合プロセッサの提供
2. 疎結合プロセッサの提供

1979年 ライブラリ機構の提供

1982年 B 7900 発表
アドレス空間番号（ASN）を用いた 192 メガバイト実メモリ空間の提供

1984年 A シリーズ発表

1986年 MCP/AS（32 ビットアドレッシング－24 ギガバイトの実メモリ空間）発表
1. ミラーディスクの提供
2. メモリディスクの提供
3. ディスクキャッシュの提供

1987年 InfoGuard（安全保護 C 2 レベル）の提供
セマンティック型データベース（SIM）の提供

1988年 A 17 発表
RMM（Resource Management Module）の採用による MCP 機能の一部をハードウェアへオフロード

1989年 MicroA 発表
1 チップ CMOS プロセッサの採用

1991年 A 19 発表
スーパースカラーアーキテクチャの導入

1992年 リモートデータベースバックアップ（RDB）の提供
関係データベース（SQLDB）の提供

1993年 A 7 発表
1. CCE/CCP（協調コンピューティング）の提供

1994年 A 18/A 14 発表
1. POSIX の支援
2. アプリケーションハートビートの提供
3. ディスクファイルのバッファ共用機能の提供

次章以降に，いくつかの主要な主題ごとに，現在まで MCP で実現されてきた新機能および技術について述べる．

2. タスク制御技術

A シリーズのタスク制御の機構において本質的なものは，プログラムを並列実行するための機構としてのスタックである．このスタック機構は，今日の大規模トランザクション処理における 1000 を越えるプログラムの並列実行を無理なくこなす技術上の基盤を作った．したがって，この 30 数年前のアーキテクチャ上の選択は，当時のメインフレームの処理性能の規模に比較し，数百・数千倍の規模をこなす今日の A シリーズにとって妥当であったといえる．本章では，スタックアーキテクチャを採用した

Aシリーズにおける多重プログラミング，多重プロセッシングの実現から，MCPを軽量化するために採用されたオフロードエンジンおよび大規模トランザクション処理を実現するためのライブラリ機能について述べる．

2.1 論理プロセッサの制御技術

2.1.1 論理プロセッサ実現の基礎技術

Aシリーズでは，すべての業務プログラム，制御系プログラム（データベース制御やトランザクション制御など）は，MCPを外側ブロックとするMCPの中の一つの手続きとして位置づけられた．すなわち，プログラムの実行はMCPによる手続き呼び出しとして実現される（図1）．この手続き呼び出し/手続きからの戻りとそれに伴う参照可能なアドレス空間の切り換えはENTERあるいはEXITというただ一つの機械語によってAシリーズのプロセッサ内で実現されている．すなわち，Aシリーズにおける多重プログラミング環境とは，MCPが自分で定義した手続き（業務プログラム等）を非同期に複数呼び出し，参照できるアドレス空間を切り換える技術で実現している．呼び出された手続き（業務プログラム）には，専有のメモリ空間が割り当てられ，このメモリ空間は，MCPを「親」（すなわち大域変数を持つ）とするアドレス参照関係が，ENTER命令によって確立される．

MCPから起動されたプログラムは，スタックという構造体で管理される．これは，手続き呼び出し/戻りの動作が，スタックという「後入れ先出し」の動作に対応するからである．Aシリーズのスタック機構では，タスク（実行単位，業務プログラムの実行によって生成されるプロセス）は，プロセススタックとセグメント辞書スタックという二つのスタックにより構成される．プロセススタックは，単純変数，構造を持ったデータ（配列など）を指し示すデータ記述子，その他の制御語を格納し，タスクの実行につれ内容が変化する．セグメント辞書スタックはアルゴリズムを実行する機械語の並び（コードセグメント）を指し示すセグメント記述子の集まりである．このようにタスクを時間と共に変化するプロセススタックと，時間と共に変化することのない（変化してはならない）セグメント辞書スタックという二つに分離し，また，この二つのスタックのアドレス空間には，プロセススタックの「親」（プロセススタックの大域変数）としてセグメント辞書スタックが関係づけられる．したがって，業務プログラムはリエントラント（再入可能）で実行可能となる．リエントラントは，セグメント辞書スタックをアドレス空間として共有する複数のプロセススタックの実行形態である．

プロセス切り換えは，切り換えたいタスクのプロセススタック（論理プロセッサ）を物理プロセッサに割り当てることで実現している．このプロセス切り換えは，ENTERとEXITによるアドレス空間の切り換えと，実行主体の切り換えという二つの機能を持ったMOVE TO STACKというただ一つの機械語により実現しているので軽量なプロセス切り換えが実現されている．

このようにシステム全体を階層化したアドレス構造にすることによって，自分の手続きを呼び出すことと全く同じ制御方法を用いて，多重プログラミング環境，リエントラントの実行環境，軽量なプロセスの切り換えをアーキテクチャレベルで実現しているのである．

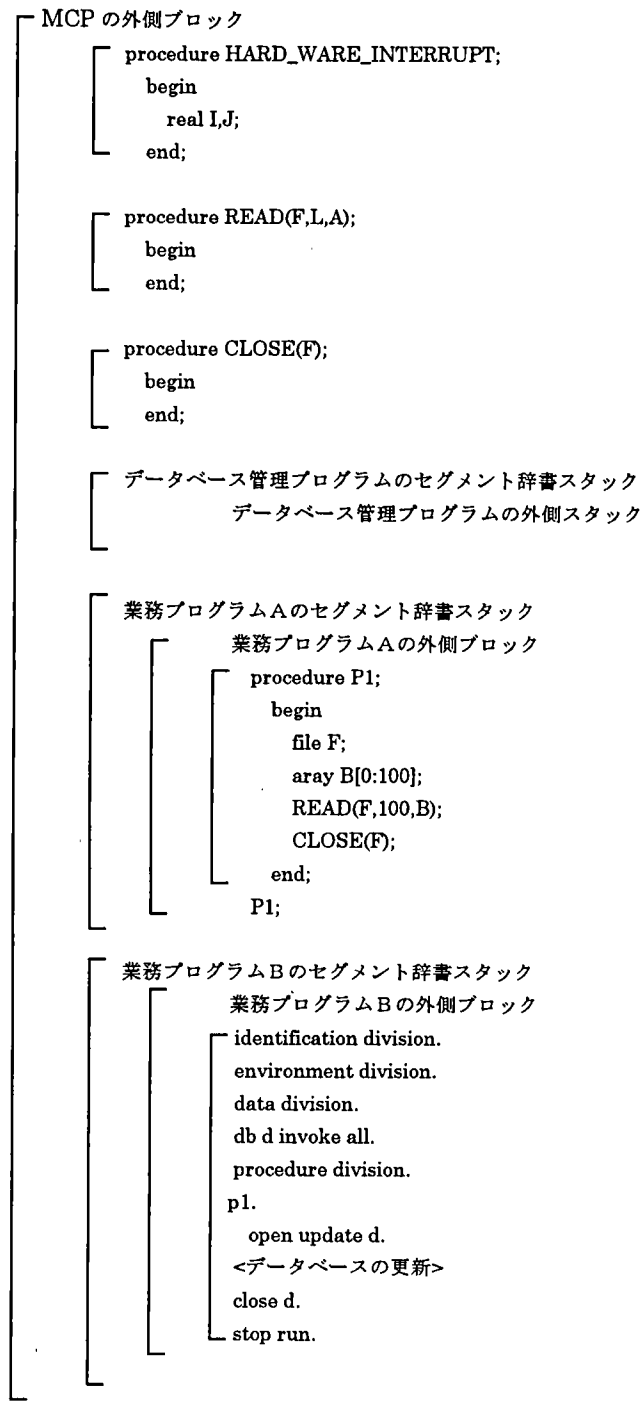


図 1　A シリーズにおけるアドレス構造

2.1.2 多重プロセッシング

MCPは1964年にマスタ・スレーブ方式の多重プロセッサの支援を開始し，さらに，1970年には多重プロセッシング方式をマスタ・スレーブ方式から対称型多重プロセッシング(SMP)方式へと変更した．このSMP方式への変更により，MCPも含め任意のタスクはどのプロセッサに割り当てられてもよいこととなり，プロセッサ資源をより有効に使用することができるようになった．SMP方式の場合，任意のプロセッサが停止しても，冗長性を利用した縮退運用が可能となり，信頼性を向上させることができる．

2.1.3 タスキング（マルチスレッディング）

1970年，従来MCPだけで実施してきたタスキングの機能（すなわち，多重プログラミングの実現機能）をプログラミング言語の構文レベルで一般プログラムに対しても公開した．これにより，システムプログラムだけではなく一般のプログラムにおいても子タスクの生成および制御等が可能となった．子タスクとは，プロセススタックを，言語のブロック単位で分割してメモリに割り当てたプロセススタックを言い，親のプロセススタックとの間で，アドレス参照関係を持つ．したがって，一つのプログラムは，同時に複数の命令コードを実行できることになる．この機能により，あるタスクが他のタスクを制御する事が可能となっただけでなく，大域変数による複数タスク間の資源の共有が可能となった．

このタスキング技術は，UNIX[*1]プロセス生成で用いられるfork操作と類似の機能ではあるが，fork操作と異なり親のスタックの内容をコピーする事はないので，プロセスの生成に余分な負荷はかからない．また，プロセス切り換えの軽量化を図るために導入されたUNIXの後継OSなどに見られるマルチスレッド機構は，アドレス空間を共有することによってオーバヘッドを減少させる機構であるが，先に述べたようにAシリーズでは，すべてのタスクがスレッドと同様に軽量のタスク切り換えを行える．

2.2 プログラム間インタフェースの制御

プログラム間インタフェースにおけるデータの受け渡しは，システム全体を一つのプログラムとみなし，階層化したアドレス構造を採用することで，安全にかつ効率的に行うことができる．一般にプログラム間のインタフェースは，割り込みを介して行ったり，待ち行列を用いたメッセージの授受等により行われている．Aシリーズでは，手続き呼び出しの機構を用いて，MCPとインタフェースし，また，1970年に支援が開始されたデータベースシステムDMS IIにおいて，業務プログラムとデータベース管理システム(データベーススタック)間のインタフェースとしてMCPへの手続き呼び出しと同じ技術が採用された．これにより，MCPのみならずDMS IIのような制御系ソフトウェアが，業務プログラムの数だけ並列実行できる機構が整った．

2.2.1 MCPとのインタフェース

業務プログラムとMCPのインタフェースは，スーパバイザコールのような割り込み方式をとらない．図1からわかるように，MCPの手続きは，業務プログラムの大域宣言と位置づけられており，したがって，業務プログラムは，入出力動作やメモリ割り当て要求を行う時，手続き呼び出しのインタフェースを用いる．これは，自分のプ

ログラムの中に定義した手続き呼び出しと全く同じに扱われる．このインタフェースを任意のプログラム間で可能にしたのが，以下に述べるライブラリ機能である．

2.2.2 ライブラリ機能

大規模なトランザクションシステムに要求されるタスク制御上の要件として，トランザクションそれぞれに対応したタスクをオペレーティングシステムあるいはハードウェアのオーバヘッドを最小限に押さえ，しかもタスク間あるいはオペレーティングシステムとのインタフェースを高速にかつ簡単な機構で行えることがあげられる．1979年に従来データベースシステムのようなシステムソフトウェアに対してのみ提供されていたライブラリ機能が一般業務プログラムに対して公開されたが，この機能は，このような要件を満たすタスク制御機能である（図2）．

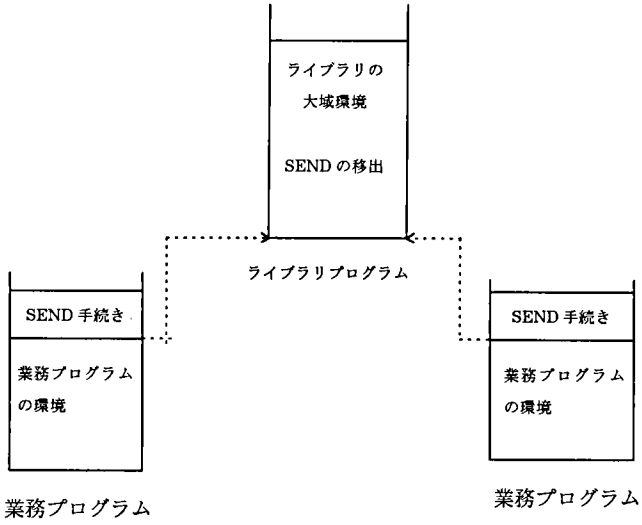


図2 ライブラリと業務プログラムの関係

ライブラリはそれ自身一つの完全なプログラムなので，自分の大域変数を持てる．したがって，ライブラリの手続きを呼び出す複数のプログラムが，プロセス切り換えという負荷なしに，ライブラリの手続きを介して他環境（ライブラリ）下の大域変数を共有（参照/更新）する事も可能である（1995年に手続きを介さなくとも直接ライブラリ内の変数を参照/更新できるようになった）．これからわかるように，ライブラリは単なる共通ルーチンの集まりというだけでなく，タスク間での資源の共有や排他制御，さらにタスク間通信の手段としても利用できる．この機能を利用することにより，各モジュールを「ブラックボックス」化し，プログラムのモジュール性を高めることが可能である．以下にライブラリの例（図3，4）を示す．

現在のシステムソフトウェアの多くがライブラリ機能を使用しているが，最も効果を発揮しているのが通信管理ソフトウェア（COMS）であろう．このソフトウェアは，トランザクション処理用の通信ソフトウェアでそれ自身ライブラリである．このソフトウェアが出現するまでのトランザクション処理では，トランザクションの受信は通信ソフトウェアが行い，それを待ち行列等の方法で業務プログラムへ渡していた，送

```
begin
  event ev;
  array table[0:100],
        export_array[0:10];;
  real x;
  procedure P(pos,val);
  begin
    procure(ev);
    table[pos]:=val;
    liberate(ev);
  end;
  export P,                          %手続き P の移出
         export_array (readwrite),   %配列 export_array の移出（読み込み/書き込み可）
         x (readonly);               %実変数 x の移出（書き込み不可）
  freeze(permanent);
end.
```

図 3　ライブラリプログラム（OBJECT/LIB）の例

```
begin
  library lib(title="OBJECT/LIB.")
    [array export_array(readwrite);
     real x(readonly);]
  procedure P(I,j);
    real I,j;
    library lib;

  P(1,100);
  export_array[x]:=1000;
end.
```

図 4　ライブラリ呼び出しプログラムの例

信についても同様である．この方法を用いると一件のトランザクションを処理するために数回のプロセス切り換えが発生することになり，トランザクションが多くなるに従いシステムのオーバヘッドが増加する．COMS を使用した場合には，トランザクションの送受信は業務プログラムから呼び出された COMS の手続き中で行われるため，従来のようなトランザクションの送受信のためのプロセス切り換えが発生しない．さらに，従来のように一つの通信ソフトウェアが全トランザクションの送受信を行わず，各業務プログラムのスタック上で送受信を行うため，通信ソフトウェアが隘路と

なって稼働できる業務プログラムの本数が効率上制限されることはなくなった．これによって，今日の大規模トランザクション処理に耐えうるプログラムの並列実行が可能となった．

2.3 タスク制御のオフロード

システムの処理能力を向上させるには，プロセッサを高速化することが当然必要である．しかし，入出力制御，マルチプログラミングやデータベースの処理のための事象管理・排他制御・タスク制御などの処理もプロセッサに行わせると，本来ユーザのアプリケーション・プログラムに割り当てるべきプロセッサ資源を浪費してしまう．今後ますます要求される非常に大規模なトランザクション処理では，多数のタスク制御と大量の入出力制御の処理が必要となり，飛躍的にオーバヘッドが増加することが予想される．

Aシリーズはその誕生以来この問題に焦点を当て，タスク制御処理を行う専用プロセッサとして独立したタスク制御プロセッサを1988年に発表したA17から採用した．タスク制御プロセッサは，タスク状態の管理とプロセッサディスパッチの制御を行う独立した専用プロセッサである．このタスク制御プロセッサの導入により，従来オペレーティングシステムが行っていたプロセススケジュール，プロセスディスパッチ，プロセス統計情報収集と管理，事象管理，排他制御の各機能はすべてプロセッサ負荷から取り除かれた．

3. メモリ制御技術

Aシリーズにおいては，その原型となったB5000（1964年発表）以降，可変長のセグメントを単位とする「動的メモリ割り当て」によるメモリ管理に基づいた仮想メモリを実現してきた．それ以来，当時の32K語（192Kバイト）の主メモリの容量は，数十メガ，数百メガ，そして現在の24ギガバイトの容量まで拡大されてきた．ここでは，セグメンテーション方式に基づく仮想メモリの技術をベースに，桁違いに拡大されてきた主メモリ容量を，オペレーティングシステムの観点から，どんな技術を用いて制御してきたかを解説する．

3.1 動的メモリ割り当て

Aシリーズは，伝統的に「セグメント」というプログラム言語の定義する論理的なかたまりを制御の対象としてきた．これは，設計当初から，プログラムの実行に対する意味モデル（プログラム言語の持つセマンティック）をコンピュータシステムの中にどう実現するかという命題において，当時のバロースのアーキテクトが選択したものである．したがって，セグメントは，配列や構造体のかたまり（データセグメント）や，セクション，サブルーティン，手続きなどの単位での命令コードのかたまり（コードセグメント）を示す．このセグメントに対して，それのアドレスや長さ，データの型，アクセス方式などの属性を定義する制御語（セグメント記述子）が割り当てられる．プロセッサは，この記述子を介してセグメント内のデータをアクセスし，メモリ上では，可変長のセグメント単位にメモリを割り当てる．

あるセグメントを必要とした時，割り込みを契機にMCPは，そのサイズに必要なメモリを確保し，記述子に確保されたメモリアドレスを設定する．可変長サイズを扱う

この方式を，動的メモリ割り当てと呼ぶ．動的メモリ割り当てを実現するには，未使用領域も可変長となるため，固定長でメモリ管理する方式にはない，解決すべきメモリ管理上のいくつかの技術的な課題があった．

第1番目の課題は，いかに速く要求されたサイズを満足する未使用メモリを探すか，である．このためにMCPは，未使用領域間をリンクさせるメモリ管理を行い，リンクの保守性をあげるため，双方向の2系列のリンクを維持した．プロセッサには，必要サイズをパラメータに，このリンクをたどって満足するサイズの未使用メモリのアドレスを探すLLLU（Linked List Lool Up）という命令コードを実装した．未使用領域のリンクには，サイズ順にリンクさせるBF（Best Fit）方式と，未使用になったメモリを常にリンクの先頭に繋ぐFF（First Fit）の二つの方式がある．システムを立ち上げた直後は，MCPの使用部分を除き，一つの大きな未使用領域がこのリンクに登録される．

第2番目の課題は，メモリの利用効率を下げるチェッカボード（フラグメンテーション）への対応である．これは，メモリ全体としては未使用領域があるが，満足するサイズの連続した未使用メモリがないという問題であり，これを解決するいくつかのアルゴリズムがMCPに実装された．

一つは，メモリ全体で，常駐用（ある処理が終わるまで同じメモリ領域を専有するセグメント用）の部分と，非常駐用（いつでも二次記憶装置に書き出せる，あるいは，メモリの別の場所に動かせるセグメント用）の部分に自然と分かれてメモリを使用させるアルゴリズムの開発である．これは，未使用領域の管理を，前後にある使用領域が，常駐型，非常駐型のいずれで使用されているかによって，常駐用と非常駐用の二つのリンクで管理することをベースに実装された．現在，常駐用にはBF方式を，非常駐用にはFF方式をもちいている．

二つ目は，ガーベジコレクションのアルゴリズムの開発である．これには，未使用領域を登録するとき，前後の領域で未使用なものがあれば合体することと，使用セグメントをメモリ内で移動させることで対応した．メモリが高い負荷で使用され出すと，大きなセグメントを満足する連続したメモリを確保できない場合がある．このケースにおいて，MCPは，いくつかのセグメントを他のメモリ領域に移動させ，大きなセグメント用のメモリを確保する．これらは，ガーベジコレクションとして機能する．

このような可変長サイズを扱うメモリ管理は，言語のセマンティックをプロセッサと協同で，高い信頼性のもとで実現しようとしたものであったが，当時の数百Kバイトの主メモリ空間を対象としたものであったのも事実である．現在の，数十ギガのメモリに対し，未使用領域を管理するリンク方式と，それを辿って探すオーバヘッドは無視できない．そこでは，さらに技術的な改良が必要となった．この問題は，3.3.3項「ASDメモリアーキテクチャ」で触れる．

3.2 セグメンテーション方式による仮想メモリ

仮想メモリは，アドレス機構としては，主メモリを，容量としては，二次記憶の補助メモリを扱うという，二段階のメモリ構造を一次元メモリとしてプログラムに見せる技術である．したがって，プログラムからは補助メモリの存在は消え，「(実用上)無限の」メモリを使用できることになる．この技術を実現するには，アドレス変換機

構と，(主メモリが補助メモリより小さいわけだから)置き換え機構が必要になる．置き換え機構は，主メモリと補助メモリ間での置き換えアルゴリズムと，置き換えの契機となるフェッチアルゴリズムから成る．

3.2.1 アドレス変換機構

Aシリーズのアドレッシングは，スタック内アドレッシングと，スタック外の記述子を介したアドレッシングの二通りある．スタック内アドレッシングは，命令コードにおける相対アドレスとしてのアドレス対（ブロック構造における入れ子のレベル，すなわちネスティングのレベルと，そこからのオフセット値の対）が絶対アドレスに変換される．プログラムの実行中は，そのプログラムの参照できる入れ子のレベルごとに基底の絶対アドレスをプロセッサのレジスタに絶えず保持しており，その値にオフセット値を加えることで絶対アドレスへの変換が行われる．スタック外アドレッシングは，先に述べたようにセグメントがメモリに割り当てられたとき，記述子に絶対アドレスが設定される．すなわち，記述子がアドレス変換機構を提供する．

3.2.2 置き換え機構

MCPは，置き換え機構として「要求時オーバレイ」と「先取りオーバレイ」の二つの方式を採用してきた．メモリへの割り当て要求は，割り込みを介してMCPが処理する．この場合，未使用メモリ量が十分存在していないシステムでは，メモリ確保を契機に他のメモリ領域をオーバレイ用のディスクに書き出したり，メモリから削除したりして，要求されたメモリを確保する，これを「要求時オーバレイ」と呼ぶ．B 5500以来，B 6700/B 7700の当初のMCPは，要求時オーバレイを実現し，古いセグメント順にメモリから追い出していた．

これに対して，「先取りオーバレイ」は，Peter J. Denningのワーキングセットモデルに基づいたものである．プログラムは，ある間隔の時間幅において比較的小さなサブセット内で参照を行う傾向にある，すなわち，ある時点で使用するメモリ量は，そのプログラムのために使用されているメモリの比較的小さい部分にすぎない（プログラムの局所性）という振る舞いをする．したがって，あるプログラムの使用しているメモリ量は，必ずしもそのままの容量分をメモリに置いておく必要はないことになる．このオーバレイ技法では，MCPが多重プログラミング状況の中でメモリを定期的に走査し，一定量のメモリをオーバレイディスクへ書き出したり，メモリから除去して未使用メモリを事前に確保する．したがって，プログラムからのメモリ確保要求時において，メモリとディスクとの間で行われる置き換えの頻度を軽減することができ，結果として各プログラムのメモリオーバヘッドに伴う経過時間を短縮することができる．

このような仮想メモリの実現技術は，やはりメモリの大規模化と高速処理の要求からさらにいくつかの技術的な改良を必要とした．今日のAシリーズの原型となった上記の仮想メモリの実現技術が，どのように改良されてきたかを，次の節で解説する．

3.3 メモリアーキテクチャの変遷

メモリの大容量化要求に対応して，Aシリーズは様々なメモリアーキテクチャを支援してきた．以下にAシリーズで支援されてきた主なメモリアーキテクチャを紹介する．

3.3.1 単一アーキテクチャ

これはAシリーズで使用された最初のメモリアーキテクチャであり，単純明快であった．前節までに述べてきたメモリ管理方式や仮想メモリは，この時代に開発されたものであり，B 5500での15ビット，B 6700/B 7700での20ビットのワードアドレッシングによる最大6メガバイトの実メモリの容量を扱っていた．

プロセスが必要とするメモリ容量が増え，かつプロセスの多重度が増大するにともない，70年代中頃から，数百メガバイトのメモリの実装の時代に入った．

3.3.2 ASN メモリアーキテクチャ

単一アーキテクチャにおける最大メモリ容量は，20ビットアドレッシングによる1メガ語（6メガバイト）という問題を解決するため，次にアドレス空間番号（ASN-Address Space Number）メモリアーキテクチャが開発された．このアーキテクチャの基本は，20ビットアドレッシング方式を変えずに，すなわち，プロセッサのアドレス関連のレジスタのビット幅の変更や利用者プログラムの再コンパイルなしに，システム全体で搭載できるメモリ量を，192メガバイトまで増やすことであった．

このASN方式の原型は，1976年のB 6800/B 7800での密結合プロセッサの開発にある．密結合プロセッサでは，メモリは，プロセッサで共用される共有メモリと，そのプロセッサだけがアドレスできる局所メモリから構成される．したがって，自分の局所メモリと共有メモリの合計が6メガバイト以内であればよく，システム全体としては6メガバイト以上のメモリを利用できる．この密結合方式では，密結合できるプロセッサの数からして，数十メガバイトがメモリの最大となる．

ASN方式は，プロセッサと局所メモリの1対1の関係を持つ密結合プロセッサ方式に対し，メモリをプロセッサの数に依存させず，n個の空間に分割し，1対nの関係を実現したものである．メモリ分割では，システム上の全実メモリを複数の「メモリ構成要素」と呼ばれるメモリ領域に分割する．一つのメモリ構成要素を「共有メモリ構成要素」に割り当て，他を「局所メモリ構成要素」に割り当てる．この各々の「局所メモリ構成要素」にアドレス空間番号と呼ばれる番号が割り当てられる．プロセッサは，このアドレス空間番号を用いて，参照するメモリ空間を切り換えながら使う．

「共有構成要素」と一つの「局所構成要素」を組み合わせて6メガバイト以内の一つの「アドレス空間」を構成する．メモリ管理におけるメモリの割り当て方式や仮想メモリの実現は，単一メモリアーキテクチャにおけるものと同じであり，それらの技術は，共有と局所の各メモリ構成要素単位に適用された．「共有構成要素」は全てのアドレス空間に存在するので各アドレス空間から共有できる．ASNシステムにおけるアドレス空間を図5に示す．

このASN方式により6メガバイト以上の実メモリの搭載が可能となったが，次のような欠点を持っていた．大部分のMCPの機能は「共有構成要素」上で実行され，かつ任意の「局所構成要素」上のタスクから参照されるもの（たとえば，データベーススタック）は，「共有構成要素」に割り当てられなければならない．そのため，「共有構成要素」の容量はある程度大きい必要があるが，大きすぎると反対に「局所構成要素」部分が小さくなり，そこではスラッシングが発生する可能性がある．このため，「共有構成要素」の容量を随時調整する必要があった．

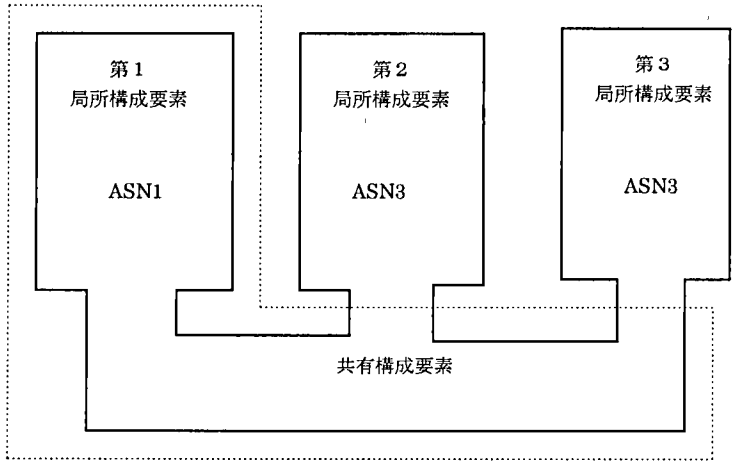


図 5　ASN システムでのアドレス空間

3.3.3　ASD メモリアーキテクチャ

1986 年に，実セグメント記述子（ASD-Actual Segment Descriptor）メモリアーキテクチャを開発し，MCP/AS（MCP 拡張システム）によって提供した．これはアドレッシングに関わるレジスタのビット幅を 32 ビットに拡大し，24 ギガバイトの実メモリ空間の実装を可能にするとともに，プログラムの修正や再コンパイルなしに，この巨大空間を使わせようとするものである．

この方式は，従来の記述子が，直接，絶対アドレスを指していたのに対し，それを ASD と呼ぶ構造体を指させ，ASD が 32 ビットのアドレスを持つというものである．したがって，絶対アドレスを指す記述子を，「実セグメント記述子（ASD）」といい，コンパイラが割り当てた従来の記述子を「仮想セグメント記述子（VSD—Virtual Segment Descriptor）」と呼ぶ．VSD から ASD への関係付けは，プロセッサと MCP とによって実行時に行われるため，従来のメモリアーキテクチャ上で実行していたプログラムは，命令コードの互換をもったまま実行できるのである．アドレス変換機構は，この ASD アーキテクチャにより，スタック内のデータを単純にレジスタのビット幅の拡大で実現する．またスタック外のセグメントへのアクセスは，VSD から ASD への変換と，ASD から絶対アドレスへの変換という 2 段階の変換機構を通して行われる．ASD は，4 語を 1 エントリとするテーブルであり，その概要を図 6 に示す．

実メモリ空間の巨大化に対応して，以下のような技術的な改良が必要となった．

1)　メモリ管理の改善……未使用領域の管理を，単一メモリアーキテクチャ当時のように，常駐用と非常駐用に分割しただけでは，そのリンクを辿るオーバヘッドは無視できない．MCP/AS では，常駐用，非常駐用とも未使用領域のサイズを約 650 種類のグループに分割し，その分割されたグループ内で，単一メモリアーキテクチャと同じメモリ管理と割り当てを行うことで，従来と変わらない効率性を達成した．

2)　オーバレイ用ディスクの並列入出力処理……メモリが大きくなれば，運用の仕

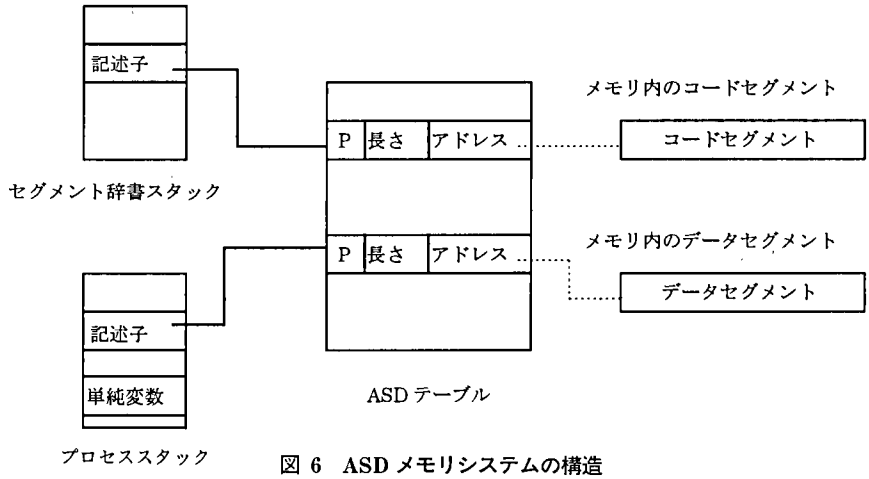


図 6　ASD メモリシステムの構造

方によっては補助記憶としてのディスクとの入出力の回数・量も増す可能性がある．従来は，オーバレイは，運用形態によって数個のディスク装置との間でしか並列に入出力処理ができなかったが，そのシステムに装備されている任意のディスク装置と並列入出力処理ができることになった．

3) 先取りオーバレイの改善……「先取りオーバレイ」に新たに「二段階オーバレイ（second chance overlay)」方式を開発した．この「二段階」と呼ばれる理由は，オーバレイの処理がオーバレイの対象となるメモリ領域を選択しマークする段階と，マークされたメモリ領域を実際にオーバレイディスクに書き出したりメモリから除去する段階の二つに分けたためである．この目的は，この二つの処理の間に一定秒の間隔をおくことにより，その間に参照されたデータは，たとえオーバレイ対象とマークされていても実際のオーバレイ処理対象から外すとことができる．これによって，使用頻度の高いセグメントを，そのプログラムのワーキングセットの中に保持していくことが可能となる．

4) アドレス変換機構の高速化……ASD 方式を実装するのにともない，二つの専用キャッシュメモリを実装した．一つは，ASD テーブルのための専用のキャッシュである．もう一つは，スタック内データのアクセスを高速化するもので，命令コードを解読するプロセッサのコードユニット内に設けられた専用キャッシュで，スタック内のアクセスされたデータを保持する．これは，通常のキャッシュメモリからのアクセスよりさらに速い．

4. 入出力制御技術

4.1 概　　要

入出力サブシステムを設計するときの最も重要な要件は，業務プログラムに対して，入出力装置の性格を意識しないですむようなインタフェースを提供し，装置の独立性を保つことである．このためのインタフェースはオペレーティングシステムによって提供される．MCP の入出力制御は，業務プログラムのみならずジョブ制御に至るま

で，装置を抽象化することで入出力操作を論理化し，さらにMCP自身も装置の制御から独立することを主要件に設計された．オペレーティングシステム自体が装置の制御から独立するという発想は，Aシリーズを柔軟性に富んだ魅力あふれるシステムにし，今日のオープン化の時代に続々と登場する，たとえば協調コンピューティングといったような新技術の取り込みを可能にしている．ここでは，MCPの入出力制御の基本となった装置の抽象化について考え方と実装技術について述べる．

4.2 装置からの独立

オペレーティングシステムが装置から独立するという考え方は，今日注目を集めているUNIXやWindows*2などの普及版オペレーティングシステムの入出力アーキテクチャの設計にも見ることができる．つまり，オペレーティングシステムが提供するインタフェースにあわせて装置ドライバをつくることで，新たな装置の接続を可能にし，オペレーティングシステム自体が装置からの独立性を保っている．このような考え方は装置の構成の変更などによる影響を受けないため，MCP同様，いわゆる“システム生成”といった作業を必要としていない．

4.3 装置の抽象化技術

業務プログラムのみならずMCPまでも装置からの独立性を保つという要件を実現するために，MCPは二つのインタフェースを提供し，装置を抽象化した．一つはFIB（ファイル情報ブロック）と呼ばれる構造体を介した業務プログラムとMCPとのインタフェースであり，もう一つはIOCB（入出力制御ブロック）と呼ばれる構造体を介したMCPと入出力プロセッサとのインタフェースである．

MCPが管理するすべての入出力資源は抽象化され，オブジェクトとして扱われる．FIBはファイルオブジェクトであり，その入出力資源のメモリ上における表現である．したがってFIBにはファイルの形式やアクセス手法，ファイルが割り当てられる装置型など，そのファイルに関するさまざまな情報が格納されている．業務プログラムが行うファイルの操作は，実際にはFIBへの操作であり，それはMCPが提供する特定の手続きを呼び出すことにより行われる．実ファイルへの関係付けはMCPが動的に行うので，業務プログラムは装置の物理的属性について配慮する必要はなく装置から独立している．

一方，IOCBは入出力要求そのものを抽象化したメモリ上の表現で，装置を直接操作するために必要な情報，たとえば装置のアドレスや入出力指令，データを格納するバッファのアドレスや長さ，入出力の処理結果などが格納される．業務プログラムが行った入出力要求に基づき，MCPはIOCBを作成し，入出力プロセッサにそのIOCBのアドレスをわたす．入出力プロセッサはそのIOCBに基づき指定された装置を操作し，処理結果を返す．

4.4 新テクノロジの実装技術

このようなMCP入出力制御の設計において採用した抽象化技術は，その後のMCPをとりまく環境の変化，とくに，高速入出力の実現やプリンタなどに見られる多種化する装置との接続，協調コンピューティングなどの異機種接続と異機種システムとの統合への要求に柔軟に対応することを可能にした．

高速入出力処理を実現するために従来から，入出力専用のプロセッサを装備してき

たが，A 17にて採用したオフロードエンジン（オフロードエンジンについては，本号「大容量データ処理を実現する入出力機構」の「機能別入出力プロセッサ」の章を参照のこと）は，高速入出力処理におけるMCPのオーバヘッドをさらに大幅に減少させた．これは，桁違いに大量化する入出力処理を，MCPのオーバヘッドを増加させることなく処理できるという画期的な技術といえる．この技術は，MCPが入出力機構から独立しているという事実なしには容易に実現できるものではなかった．オフロードエンジンの搭載により，システムの処理能力が大幅に改善された．

昨今のオープン化の流れの中で，オペレーティングシステムの入出力制御に求められるものは，ディスク，テープ，プリンタなどの入出力装置を固有のインタフェースで支援することにとどまらず，多種多彩な装置を，その場所を問わず支援できることである．そのためには，業界標準といわれる各種インタフェースを柔軟に組み入れることができなければならない．MCPの入出力制御は，入出力資源を抽象化して管理しているため，このような要求に容易に対応できる以下のような技術的適用性がある．

異機種システムとの接続においては，相手システムの持つ固有の特性を吸収し，相手システムとのプログラム間インタフェースを，自分のシステム内で行うのと同等に行えなければならない．このために，仮想装置を扱うポートファイルと，独自の制御システムを装置として扱うEXTERNAL I/O（外部入出力），装置固有の属性をMCPの入出力処理から独立させたライブラリインタフェースが基礎技術となった．たとえばポートファイルにより，互いを仮想装置と見立てることにより，複数の業務プログラム間でのプログラム間インタフェースを実現している．これらの業務プログラムは，同一ホストで稼働している必要はない．また，Aシリーズで搭載したCCE/CCP（協調コンピューティング）は，UNIXとのRPC（遠隔手続き呼び出し）を実現している．このCCE/CCPの実装技術において，MCPはUNIX機を一つの装置として扱うことで抽象化した．すなわちEXTERNAL I/Oを用いることにより，旧来の装置をアクセスするのと全く同じようにUNIX機にアクセスできるようにしているのである．

これらの技術を使うことにより，業務システムは同一システム内で扱うのと同じやり方で，異機種システムと統合した環境を構築できる技術基盤が提供されたことになる．

さらに，EXTERNAL I/Oが，物理的なインタフェースをMCPから独立して扱うのに対し，装置の特性やそれの運用管理面を扱うライブラリがMCPとのインタフェースとして提供されている．この機構をプリンタ装置へ適用し，プリンタ独自の制御コードの処理やコード変換をライブラリに行わせることにより，MCPやプリントサブシステムを変更することなく，昨今多様化しているプリンタへ対応することができる．

5. 耐障害性技術

コンピュータの登場以来，高信頼性システムへの改善は，休むことなく続けられてきた．その対象分野は，故障発生の未然防止と故障検出の高度化，障害の局所化，障害の自動排除，早く確実な障害回復などの分野である．ここでは，それらの分野について，主にオペレーティングシステム，あるいは，それが協調動作するハードウェア

機能に焦点を当て，その技術的な変遷を述べる．

5.1 故障の検出と障害の局所化

ハードウェアの物理的機能障害やソフトウェアバグなどによるデータ処理誤りを，ここでは「故障」といい，処理をそのまま続行するとデータの整合性が保証できないとき障害となる．したがって，システムを障害から守るためには，故障そのものを抑えることが基本であり，ハードウェア，ソフトウェアの信頼性向上が故障発生の未然防止となる．しかしながら故障の発生を完璧に防ぐことができない以上，故障の検出機構の改善と障害をどう局所化するかが次の問題となる．障害を局所化できると，それに影響のあるハードウェア，ソフトウェアの部分を使用不能にする，あるいは，回復機能を働かせるなどができるからである．

5.1.1 タグ・記述子方式による高信頼性

B 5000 以来，故障の検出機構に関して，データ移送やデータ加工におけるデータビット誤りの検証にとどまらず，命令コードの実行の連鎖において，データの整合性検証を行うことをアーキテクチャ上の特徴としてきた．すなわち，プログラムの誤動作や命令コードと扱うデータの不整合からシステムを防御する機構上の仕組みとして，データに意味を持たせるタグ・記述子方式を実装したのである．

タグは，各種の制御語，単精度語，倍精度語，配列型，コードを示す命令語など，データの型を示し，記述子は配列型のセグメントに対して設けられ，そのセグメントの長さや，アクセス単位(4 ビット単位やバイト，語単位など)，読みとり専用か否か，等などのデータの属性を示す．したがって，命令コードにより扱われるメモリ上のデータは，命令コードの操作に対して，その妥当性の可否を「自己主張」する事が可能となる．したがって，演算命令で制御語やコードをデータとして扱えないし，配列や構造体型のデータの索引において，その範囲を超えてアクセスすることもできない．さらに，コードのような読み取り専用のデータに書き込むこともできない．このような検証が，命令コードのプロセッサ内部の各実行のステップで行われる．このようなデータの誤用はすべて，MCP への例外処理として報告される．

5.1.2 MCP の割り込み処理

上記のようなデータの誤用による MCP への例外処理は，割り込みとして報告される．この割り込み機構は，A 16 シリーズの提供とともに従来のやり方を変えた．1970 年の B 6700/B 7700 以来，MCP の割り込みの入り口点は一つであり，プロセッサは，割り込みの種類と，付加情報をパラメタとして MCP に渡す．MCP はそれらを分析し，割り込みの詳細を確定し，対応処理を行う．しかしながら，この方式はハードウェアの高度化にともない，ハードウェアがつくるパラメタの複雑さが増し，当然，それを解析する MCP のロジックも複雑なものとなってきた．したがって，この方式を止め，割り込みの入り口点をその種類に対応してベクトル形式で扱うようになった．これによって割り込みの種類の追加や削除，変更に容易に対応できるようになった．

5.1.3 スタックによる参照アドレス空間の制限

システム全体をブロック構造を持った巨大な一つのプログラムとみなし，スタックを用いて多重プログラミング環境を実現していることを，第 2 章で解説した．そこでは，あるプログラムの命令コードの実行の時々において，言語のセマンティックで認

められた範囲でのみ，メモリアクセスが許される．したがって，ある業務プログラムがアクセスできない他のプログラムの専有するメモリ領域へのアクセスは禁止され，「故障」による障害の影響は，それに関連したスタックにのみ局所化される．

障害の影響が，それを引き起こしたスタックにのみ局所化されると，プログラムに，その例外事象から回復する手続きを記述させる手段の提供も行える．特に，制御系のソフトウェアが例外事象の通知（割り込み）を受け，それへの回復手段を，実行している処理ロジックに基づいて記述する事によって，システム全体を制御するMCPや，トランザクション処理システムやデータベース処理システムを，「障害に強い」ものにしていくことができる．MCPの開発以来，この改良が中断されることなく継続されている．たとえば，A 16シリーズの発表とともに，MCPの扱うシステムに大域の資源の排他制御において，排他制御したまま例外事象を検出すると，システム停止を回避するような回復ロジックが部分的ではあるが改良されてきた．

5.2 障害の自動排除

高信頼性構造においても，やはり，なお故障は発生すると考え，この場合でもできる限りシステムの障害にいたらない機構が必要である．この分野では，自己回復機能（自動修正機能）と単純な再試行，あるいは，冗長な機能を用いた再試行が対象になる．

5.2.1 システムレベルの障害の自動排除

自己回復機能は，入出力動作のように，動作そのものをMCPと協調しながら最初からやり直すレベルから，命令コードレベルの自己回復にまで拡大しようとすると，命令コードのレベルで再試行できるように障害が局所化される必要がある．命令コードレベルの自己回復機能は，1970年のB 7700から実装された．この当時，同時にプロセッサ，入出力プロセッサ，メモリの各制御装置にフェールレジスタが設けられ，MCPは，その情報を利用してフェールソフト機能を実装することになった．

ハードウェアの致命的な故障や，MCPに再試行を委ねる故障は，フェールレジスタに記録されている詳細な故障情報とともに割り込みを介して報告される．さらに，間欠的に発生する故障は，MCPによって定期的に記録され，ある時間間隔において設定されたしきい値を越えると，「致命的な故障を持った装置」と判断する．MCPは，冗長な構成に基づいて，他の代替装置（別のプロセッサ，別の入出力プロセッサや入出力経路）を用いて再試行し，致命的な故障を発生させた機器の再使用を行わないように縮退処理を行う．

A 16シリーズの発表以降，保守プロセッサ専用のシステム制御プロセッサ（SCP-System Control Processor）に大幅な機能改善が行われ，自己回復機能の分野では，マイクロコードのパリティエラーにともなう自動再ロードやキャッシュメモリの自動修正などの機能が付加された．

5.2.2 ディスクファイルシステムの障害の自動回復

ディスクファイルシステムの障害は，システムの安定稼働の阻害要因となる．したがって，ディスクファイルシステムの安全性・安定性維持のために，ディスクファイルの多重化の機能を提供してきた．以前には，ディスク領域を絶対アドレスで割り当てさせ，その領域が物理的に故障をしない限りデータを保証するIAD（Installation Allocated Disk）機能，ディスクディレクトリやディスクファイルのジョブ制御言語

の指定による重複化の機能などがある．

ディスクファイルシステムは，1986年のミラーディスクの提供によって極めて高い信頼性を確保できるようになった．この機能は，あるディスク駆動装置のデータと全く同じイメージを，最大四つまで維持する機能である．コピーをもつディスク駆動装置のハードウェア接続を，それぞれ別の入出力プロセッサと入出力経路から行えば，さらに，その信頼性は高まる．コピーをもつミラーディスク間のデータの整合性は，ミラー関係をもつ全ディスクへの書き込み動作の完了の入出力プロセッサによる確認によって維持される．この書き込み終了の確認を終わる前に，システム停止のような事態が発生した場合，システムの再立ち上げ時に，MCPは，メモリに保持していたオーディットを用いて整合性を維持する．ミラーディスク間のデータの整合性が保証できないとき（たとえば，あるディスクへの書き込み動作で，回復不能の入出力誤りが発生した場合など），MCPは，ミラーディスク関係を破棄し，処理を続行させ，エラーを起こしたディスクを使用不能にし，操作員に報告する．操作員は，再度，別のディスクにミラーをいつでも作成する指示を出せる．

ミラー関係をもつあるディスク駆動装置がなんらかの原因で作動不能の場合，MCPは，オーディットを取り始め，処理を続行させる．そのディスクが作動可能になると，オーディットが適用されデータの一致性を確保する．

5.3 迅速な障害回復

5.3.1 システムの障害回復

システムの一部が切り離されたり，システム停止に至った場合，システム運用面への影響を最小限にするべく，充実した保守機能による早く確実な障害回復が遂行されねばならない．SCPによるADCCR（Automatic Diagnostic, Collection, Correction and Recovery）機能は，故障個所の診断，分析情報の収集，システムの立ち上げ操作を自動化し，停止時間の短縮が行われた．

5.3.2 ディスクファイルシステムの障害回復

ディスクファイルシステムの障害は，ミラーディスクの適用によって，極めて高い信頼性を確保できるが，冗長なディスク駆動装置への投資も必要になる．したがって，ディスクのファイル構造を障害に強い構造にしておくことが基本となる．一般に，ディスクファイルは，そのファイルの属性や，ディスクのアドレスなどの情報を格納する「ディスクファイル見出し」という構造体と，その見出しを集めた「ディスクディレクトリ」で管理される．ファイル名の検索効率を上げるために，階層構造にしたファイル名を木構造で管理するのが自然であるが，ファイル名とディスクディレクトリ内の見出し部との関係付けの実装にはいろいろなやり方がある．

B 6700の当初のMCPでは，階層構造にしたファイル名の各レベルごとにディレクトリを設け，ディレクトリ自身を木構造にした．各レベルのファイルの名前をハッシングし，その結果の商と余りを用いて目的のディレクトリを辿り，最後にディスク見出しをアクセスする構造をとった．この構造では，ディレクトリの一部にディスク障害が発生すると，そこに関連するすべてのファイルへのアクセスができなくなるという問題があった．

1970年代の中頃，この欠陥を除去すべくディスクファイル管理の構造を変え，障害

を局所化すると同時に自動回復の機能も取り入れ，かつ，重複化することも可能となった．ディスク見出しを格納するディレクトリは，フラットな形（ディスク見出し部は，ディレクトリ内の空いているところなら何処にでも格納できる）にし，そこへのアクセスのために木構造したアクセス構造を新たに設けた．アクセス構造には，ディスク見出しにあるファイルの名前を集め，木構造で管理し，木構造の最後の「葉」のノードが，ディスク見出しをポイントする．この構造では，アクセス構造の障害は，その報告に基づいてMCPはディスク見出しからいつでも再構築できる点と，ディレクトリの障害が局所化できる点で，従来より改善された．

データベースファイルに関しては，特別な障害回復手段が1992年に遠隔データベースバックアップ（RDB-Remote Database Backup）として提供された．地震や火災などが発生した場合，そのデータベースをもつホストシステム自体も破壊される可能性がある．このため，そのデータベースと同期した複製を，遠隔地にある別のホストシステムに準備することができる．一方のホストシステムに障害が発生して，そこのデータベースを使用できなくなったときは，複製をかわりに使用する．

5.3.3　業務プログラムレベルの障害回復

業務プログラムレベル，運用レベルでプログラムが正常に動作しているかどうかを判定できる仕組みが，1994年のA 18とともに提供された．たとえば，そのプログラムが決して生起しない事象を待機したり，無限ループに陥ったりした場合，それを検出し，回復処理を実行する仕組みである．この仕組みは，MCPが監視対象とするプログラムと，MCPが障害と判定したときに実行すべき回復処理用のプログラムを登録させ，監視対象プログラムから一定の時間間隔で特定の手続きを呼び出させる（ハートビートを行う）というものである．指定の間隔でその手続きが呼び出されないと，MCPは登録された回復処理用プログラムを起動する．

6.　オープン環境下でのMCPの意義

最後に，Aシリーズがオープン化に対応するために，現在支援している技術と将来実現されるであろう技術について述べる．

近年，ミッションクリティカルな業務をオープン環境下で行いたいという要求が増大してきている．この要求に対して，当社は高信頼性のもとに大規模トランザクション処理が行えるメインフレームと，オープン環境を備えたPCという，二つの「異質な」（Heterogeneous）環境を統合した，一つの仮想マシン（HMP-Heterogeneous Multi-Processing）を提供することによって解決しようとしている．ここで言う「統合」には，何らかの機構（たとえば高速バスやLAN）で接続された二つの環境が協調して一つの機能を実現するものと，ある一つのプラットフォーム上に異なる環境が同居する，たとえば，PCのプラットフォーム上にWindowsNT[*2]とAシリーズが共存するものの2種類が考えられる．

どちらの統合を実現する場合にも，二つの異なる環境間で稼働するプロセスが協調するためのプロセス間通信機能と，両方の環境から互いのファイルをアクセスするためのファイルの共有機能が必要となる．一つ目のプロセス間通信は，Aシリーズ上では，すでに4章で述べたポートファイルによって実現することができる．すなわち，

Aシリーズ上のプロセスは，相手環境のプロセスを仮想入出力装置と見なし，ポートを介してメッセージを送受信することにより，プロセス間通信を行うことが可能である．二つ目のファイル共用機能をAシリーズで実現する場合，Aシリーズのファイルシステムにおける PC のファイルの管理を考慮する必要がある．この PC ファイルの管理とは，言い換えれば，ファイル名を含めたファイル属性をいかに取り扱うかであるが，これはAシリーズ上ですでに実現しているファイルシステムで吸収できる．たとえば，Aシリーズのファイル名は，各節点が最大17バイトの木構造になっているので無理なく Windows のファイル名を表現できる．さらに，現在，ファイル名長255バイトの NTFS（NT ファイルシステム）への対応が検討されている．

ここまで見てきたように，ミッションクリティカルな業務をオープン環境下で行いたいという要求に対応するための機能を実現する場合にも，これまで培われた MCP の技術がそのまま矛盾なく適用されていることがわかる．

7. おわりに

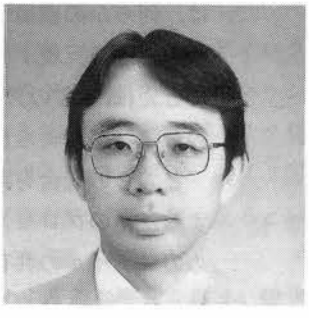
1961年の発表から現在まで，MCP が実現した主要な技術をテーマ毎に紹介してきた．Aシリーズは現在，これらの技術により障害に対する頑強性や信頼性，処理能力の向上が図られ，大規模トランザクション処理を支援するエンタープライズ・サーバに位置づけられている．さらに近い将来，HMP をはじめとする種々の技術を実装する事により，今日のオープン化の流れに対応できるオープン・エンタープライズ・サーバへと位置づけられていくであろう．これからもメインフレームに対する様々な要求がなされるであろうが，Aシリーズ，さらに MCP がその意義を失うことはないと考える．

---

*1 UNIX は，X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
*2 Windows は，Microsoft 社の登録商標である．
*3 WindowsNT は，Microsoft 社の商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

参考文献
［1］ E. I. オーガニック，土居範久訳，「計算機システムの構造―バロース大型計算機シリーズ―」，共立出版．
［2］ 高橋延匡・土居範久・益田隆司，「オペレーティング・システムの機能と構成」，岩波講座 情報科学―16，岩波書店．
［3］ 前川守・所真理雄・清水謙多郎，「分散オペレーティングシステム UNIX の次にくるもの」，共立出版株式会社．
［4］ 「Aシリーズオペレーティングシステム―構造と制御―」，日本ユニシス(株)1992．

執筆者紹介 鈴 木 秀 明（Hideaki Suzuki）
1953年生．1978年慶應義塾大学工学部管理工学科卒業．1985年日本ユニシス(株)入社．Aシリーズの基本ソフトウェアの受け入れ・保守に従事．現在，システムプロダクト部Aシリーズソフトウェア開発室に所属．



ITASCA3800 シリーズのハードウェア技術

ITASCA 3800 シリーズ・システムは，高性能な 0.5 ミクロン CMOS（相補型金属酸化膜半導体）標準セル・チップの先進ファミリの基盤技術をユニシス独自のロジックとカスタム分散 RAM アレイ技術に結び付けて実現されたものである．CMOS チップ技術やオンチップ RAM の積極導入，ハードウェア実装への革新的なアプローチにより，超小型，高性能，低コストの処理装置を実現した（写真 1）．

写真 1　ITASCA3800 シリーズ・システム

1. ITASCA3800 シリーズ・システムの特長

ITASCA 3800 シリーズは，2200 システム・ファミリにおける中規模から大規模のエンタープライズ・サーバ・プラットフォームである．価格・性能比の向上，拡張処理アーキテクチャ（XTPA）の導入，全く新しい高速の I/O，およびデータ通信構造が，この新システムの特長である．この CMOS ベースに一新されたシステムは，次のような画期的な特長を持っている．

- ITASCA 3800 エンタープライズ・サーバは，従来の C シリーズ・システムのメモリ容量の制限事項を廃した 2200/XPA のアーキテクチャを採用している．このシステムは，機能的に，2200 M シリーズと互換性があり，1100 ファミリおよび 2200 ファミリのシステムとコード上の互換性を保っている．
- 最大八つの中央処理装置（IP），最大 1 ギガ語の主記憶装置，および 12 個の入出力処理装置を，19 インチ幅のラックにマウントされた一台のシステムとして構成することができる．新設計の CSIOP（チャネル・サービス入出力プロセッサ）が，最大 192 個のチャネル・インタフェースとメモリとの間のデータ処理を担っている．

- ITASCA 3800 エンタープライズ・サーバは，XPC（拡張データ処理装置）を組入れたシステムの構築が可能である．DM（データ・ムーバ）と XPC 並列プロセッサにより，個々のシステムに共通のキャッシュが作られ，使用頻度の高い大容量のデータが保持される．これによりディスクへの入出力回数は飛躍的に削減され，スループットは大幅に向上し，データベースの復旧とオーディットの処理は高速になり，コスト効率がさらに向上した．
- ITASCA 3800 の IOP は，従来の 2200 システムよりも多数のチャネルが接続可能である．CSIOP（コモンサービス入出力プロセッサ）は入出力処理系の中心に位置付けられ，最大 12 個の CSIOP がシステム内に構成可能である．各 CSIOP はそれぞれ最大 16 チャネルをサポートでき，システム全体で 192 チャネルを接続できる。BMC，SBCON，SCSI-2 W，FDDI，Ethernet，および，ATM チャネル・アダプタがすべてサポートされる予定である．
- 入出力および通信情報関連チャネルアダプタは，現在および将来の大規模データ処理の要求に十分対応可能な設計となっている．チャネル・アダプタおよび通信アダプタがシステムに直接接続されることにより，非常に高い性能および価格比をもたらしている．これらは，新しい 2200 システムにおける通信および入出力関連でのフロントエンドの実現であり，最先端技術利用による高速データ処理要求に対応するよう設計されたものである．
- 入出力処理系は，Ethernet および FDDI チャネルをサポートして通信リンクを実現し，オープンな相互運用性を高いレベルで実現している．また，SCSI-2 ワイドチャネルが新らしいテープ装置や大容量記憶装置に対応している．さらに，既存の主要周辺装置や通信ハードウェアが接続可能となるよう BMC チャネルもサポートしている．
- この新しい入出力装置は，ATM（非同期転送モード）チャネル・アダプタにも対応している。ATM 処理に対応した新しいソフトウェアを共用することにより，ATM アダプタは，データ，図形，音声，画像のマルチメディア情報などを処理するために必要な大規模転送処理が可能となる．
- 新機能として ANSI 規格の SBCON（Single Byte Command Code Sets Connection）チャネルがあるが，これは光ファイバ技術を用いて，ブロック・マルチプレクサ・チャネルの場合より最大で 4 倍高速にデータを転送することができる．SBCON チャネルは，最大 9 km までの距離にある周辺装置の接続に対応する．
- 新しい通信アダプタおよび I/O アダプタは，並列処理についてのサポートを改善している．これらにより，密結合，重結合，または疎結合の 2200 マルチホスト構成が可能である．
- ITASCA 3800 エンタープライズ・サーバは，超巨大企業においても中央に集積された情報を管理できるよう設計されたシステムである．ITASCA 3800 の集中データ制御機能およびネットワーク管理機能により，企業全体にわたる情報を効率良く管理することが可能である．エンタープライズ・バックアップおよびリストア機能は，部間に分散したサーバやワークステーション上のデータの機密を保持することもできる．

● ユーザは，現在使用中の大部分の周辺装置サブシステムを継続使用することができる．一方，今日使われている低コストのSCSI (Small Computer Systems Interface) の周辺装置を使用できるという利点もある．レガシー(OS 2200)アプリケーションは，ITASCA 3800シリーズ・システムと完全な互換性がある．

● 新システムでは，拡張モード命令セットの利点が充分に発揮される．基本モードと拡張モードの両方で書かれたアプリケーションも，ITASCA 3800エンタープライズ・サーバ上で実行できる．基本モードのアプリケーションからこれらのシステムに移行してきたユーザは，ただちに性能の向上に気付くはずである．

2. システムの概要

ITASCA 3800シリーズ・システムは，CEC (Central Electronics Complex；中央処理装置群)，SCF(System Control Facility；システム制御装置)，周辺装置，通信制御装置，および，XPC (拡張データ処理装置) から構成されている．

CECは演算処理機能と入出力機能を持っており，ビジネス処理，科学技術計算，トランザクション処理などのタスクを実行する。CECとしては，PCC (Processing Complex Cabinet；処理装置キャビネット) およびAUX (AUxiliary Cabinet，チャネル増設キャビネット) がある．PCCには，CPM (Central Processing Complex；中央処理装置) とチャネル・ラック・モジュールがある．CPMには，インストラクションプロセッサ，記憶制御装置，主記憶ユニット，I/Oプロセッサ，システム・クロック，および，ハードウェア・サポート・モジュールがある．

1) 処理装置キャビネット (写真2，3)

本システムは，19インチ・ラック・マウント製品用の業界規格に基づいてユニシスが開発したエンクロージャに収められている．PCC (処理装置キャビネット) には36U (1U＝44.45 mm) のマウント・スペースがあり，その内上側の21UをCPM (中央処理装置) が占めている．二つのチャネル・モジュール (それぞれ5Uずつ) をPCCの下側の10Uに置くことができる．PCCの残りのスペースは，ディスク・アレイなどの周辺装置モジュールのマウントに使用できる．PCCの寸法は，548.64 mm(幅)×1099.31 mm (奥行)×1767.08 mm (高さ) であり，床面積は0.60 m^2 にすぎない (キャビネットの前後のアクセス・スペースを除く)．I/Oケーブルを配線するために，PCCの背面に304.8 mm以上の広さの囲まれたスペースが用意されている．

チャネルキャビネットはPCCと同寸法であり，内部マウント・スペースのサイズも同じ(36U)で，これを用いてCECを完成させることができる。チャネルキャビネットはチャネル・モジュールや周辺装置モジュールを追加するために用いられるが，PCCの隣に置く必要はない．

CECは空冷式であり，温度13°C～30°Cおよび湿度40%～80%と定義される「商用オフィス」環境に設置できる．必須条件ではないが，CECは，約400 mmの高さの上げ床の上に設置するのが望ましい．

2) ITASCA 3800システムの中央処理装置モジュールの構造

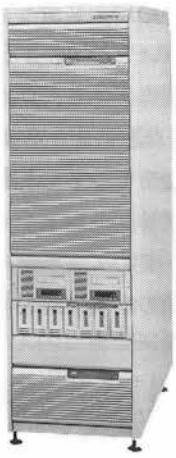


写真 2　ITASCA3800 シリーズ処理装置キャビネット

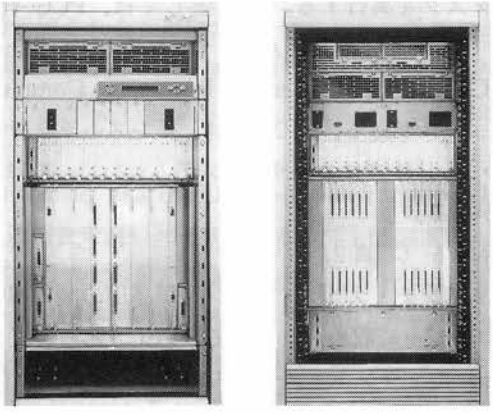


写真 3　ITASCA3800 シリーズ PCC－正面図および背面図
（パネルをはずした状態）

ITASCA 3800 システムの CPM（中央処理装置）は，プロセッサ，記憶装置，I/O，保守インタフェース機能とともに，それらが必要とする電力，電力制御，および冷却の機構を備えている．CPM は，二つの完全に独立したパワードメインからなっており，DC 電源と冷却機構については N＋1 の冗長性を実現しており，電源を入れたままの状態での交換が可能となっている．

CPM は，ユニシスが開発した標準のビルディングブロック方式(ユニシスの製品提供を通して広く使用されている）から構築されている．これらのビルディングブロックは，冷却および電力入力部，ロジック電力，そして共通した機械的また電気的インタフェースを提供し，現在および将来にわたる製品の必要条件を満たすように，すべてが高い融通性を持って設計されている．

プロセッサとコモン I/O の各々のバックパネルは，バックパネルプリント回路基板

に装着されたコネクタにて背中合わせの形で相互接続されており，ロジックと電力の相互接続の基本的な構造を形成して以下の装置をサポートしている（図1）．

- ●ドメイン当たり四つのIP（中央処理装置）
- ●ドメイン当たり512ギガ語の主記憶装置
- ●ドメイン当たり一つのネットワーク・インタフェース・モジュール（保守インタフェース）
- ●システム・クロックおよびI/Oクロック
- ●ドメイン当たり12個の独自のI/Oバス・インタフェース（一つのインタフェースはチャネル・アダプタを八つサポートできる。例えば，SBCONチャネル.）

電源および冷却機能を始めとして，本システムの重要な機能の多くは，システムに与える影響を最小限にとどめながら維持することができる．NIM（ネットワーク・インタフェース・モジュール）およびオフライン状態のMSU（大容量記憶装置）のプリント基板は，動作中のドメインでの組み込みまたは取り外しが可能である．プロセッサ/ストレージコントローラが搭載されたプリント基盤は，パワーオン状態であるが論

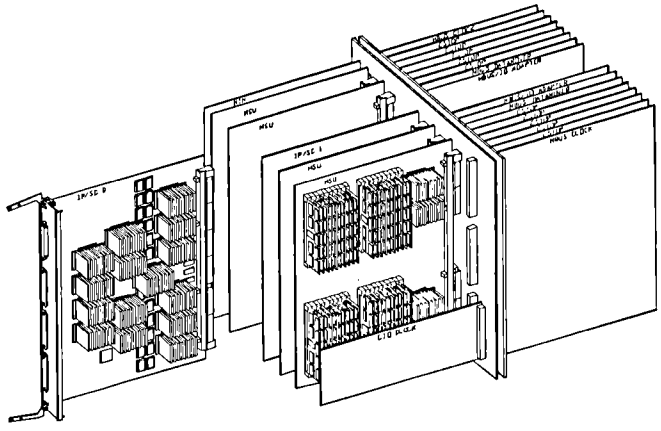


図1　ITASCA3800中央処理装置モジュールのデッキ

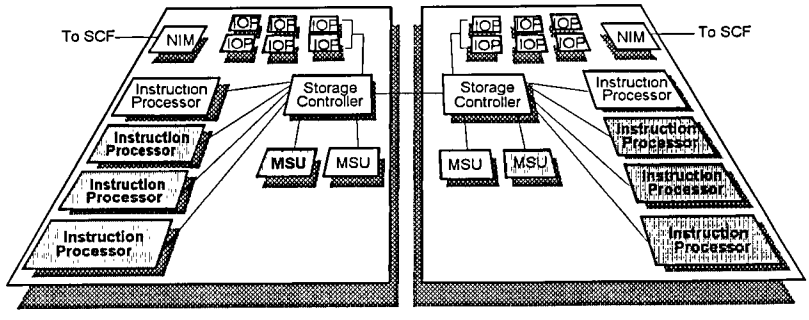


図2　ITASCA3800中央処理装置モジュールの構造

理的には他の動作ドメインからはパーティション（分離）されているドメインに組み込みあるいは取り外しが可能である（図2）.

インストラクション・プロセッサ（IP）は，2個のCMOSチップによって実現されている．一枚の基板上に最大四つのIPと1個のストレージ・コントローラを搭載することができる．ITASCA 3800システムは，スタイルに応じて1システム当たり1個から8個のインストラクション・プロセッサを持つことができる．IPには8K語のキャッシュが二つ組み込まれており，一つはオペランド用で，もう一つは命令用のキャッシュである．

ストレージ・コントローラ（SC）は，IPおよびI/Oシステムとメイン・メモリの間に位置し，2レベルのキャッシュ間の結合の緊密性を維持している．ITASCA 3800では，システム内に最大二つのSCがある．SCごとに，最大四つのIP，一つのI/Oシステム，また別のSCを接続することができる．SCには，二つのキャッシュから構成されている2次キャッシュ（SLC）がある．一つはIP命令専用で，もう一つは共有オペランド専用のキャッシュである．SCには，512K語の命令キャッシュ，112K語のオペランド・キャッシュがある．これらはアドレスにより四つのセグメントに分割されている．

デュアル・ドメイン・システムでは，1メガ語の命令キャッシュおよび224K語のオペランド・キャッシュが構成されている．最小構成のシステムには64メガ語（256メガバイト）の主記憶があり，64メガ語（256メガバイト）の単位で最大1ギガ語（4ギガバイト）まで拡張可能である．

3) 0.5ミクロン，高密度CMOSチップの技術

ITASCA 3800シリーズの新しい技術のうち最も重要なものは，高密度，低コストのCMOS標準セル・アレイである（写真4）．このチップは，標準セル設計に関連した高性能機能を最大に利用して設計されている．システムには15.5 mm角と9.95 mm角の二つのダイ・サイズがあり，それぞれ最大で800万および160万個のトランジスタ

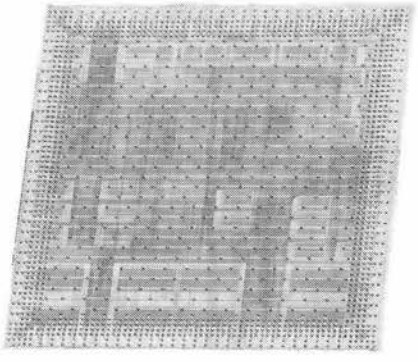


写真4 オンチップRAMをサポートしている標準セル・ロジックとカスタム・アレイを持つITASCA3800シリーズのチップのダイ

を搭載している．

高信頼，高性能，低コストな，CMOS ASICの製造に長年にわたる実績があることから，IBM社がユニシスのASICチップの提供の技術パートナーとして選ばれた．CMOSチップは，IBM社標準のCOMS 5L技術によって構築されている．プロセスは，nおよびpチャネル・デバイスの両方を用いて，0.46 μの実効NFETチャネル長および0.51 μの実効PFETチャネル長の性能を実現している．

両チップ・サイズとも，4層または5層の金属相互接続に対応しており，またIBM社のC4バンプ・ダイ接続技術にも対応している．内部セル接続は，主として金属層1とポリ・シリコンで行われている．セル間接続は，主として金属層2，金属層3，および一部では金属層4で行われている．電力供給は，主に金属層4および5で行われている．

チップの動作電圧は3.3 Vで，通常の入出力も3.3 Vで行われる．ただし，特殊なI/Oセルは5.0 Vの入力レベルを受け付けられるようになっている．電力が低下した場合にも駆動できる特殊なI/Oセルも搭載されている．

ゲートの相互接続は短いものであるが，寄生キャパシタンスはゲートの遅延を大幅に増加させる．設計を最適化するために，多くのセルが四つまでのゲート負荷に対応できるように作られており，相互接続の遅延を最少にするような出力となっている。拡張セル配置も利用されて，内部接続を短くしている．

CMOSには，DCとACという2種類の電源がある。AC電源はCV^2Fに比例している（ここで，C＝キャパシタンス，V＝電圧，F＝周波数）．ITASCA 3800シリーズでは，15 mmチップの平均キャパシタンスは14 nファラッドであり，消費電力（AC＋DC）の代表値は15 Wである．

チップを収納するパッケージには，613個のパッケージ・ピンが付いている．そのうち468個が入出力用のピンとして使用される．残りのピンは電力供給と接地のために使用される．

表1は，ITASCA 3800シリーズのIP設計で使用されているASICを2200/500シ

表1　ITASCA3800で使用されているCMOS ASIC

	ITASCA 3800	2200/500
ダイ・サイズ	15.5 mm×15.5 mm	15 mm×15 mm
金属層	5層	3層
線幅	0.49ミクロン	0.70ミクロン
消費電力	1.2マイクロW/ゲート/MHz	3.0マイクロW/ゲート/MHz
伝達遅延(代表値)	250ピコ秒	365ピコ秒
最大同等ゲート数	2,000,000	318,000
実際に使用される平均同等ゲート数	441,000	39,000
実際に使用される平均RAMビット数	783,000	152,000
IP当たりのASIC数	2	18

ステムの設計で使用されているASICと比較したものである．

4) オンチップ・メモリの採用

インストラクション・プロセッサのメモリは，すべてチップ上に載っている．キャッシュ・タグ・メモリ，キャッシュ・メモリ，制御メモリ，GRS（汎用レジスタ・セット）などのインストラクション・プロセッサにおけるメモリ機能は，メモリやそれを取り巻くロジックが同じチップ上に載っていることから，大きな恩恵を受けている．このことによって，それらが別々のチップ上に載っている場合にセクション間で起きる遅延をなくすことができる．ITASCA 3800シリーズは，RAM（随時読み書き可能型記憶素子）に3種類の独自のメガセルを，ROM（読み取り専用記憶素子）には一つのメガセルを使用している．これらのメモリの特性を表2に示す．

表2 ITASCA3800のメモリ特性

RAMの種類	語数	1語当たりのビット数	メガセル当たりのビット数	読み取りアクセス時間(ns)	読み取りまたは書き込みサイクル(ns)
RAM 256×40	256	40	10240	5.5	11
RAM 1K×40	1024	40	40960	6.5	11
RAM 2K×44	2048	44	90112	7.4	11
ROM 2K×40	2048	40	81920	7.25	22

IPおよびSCでは合わせて4種類のASICが使用されており，全体で95個の組み込みメガセルが使用されている．これらのチップにおけるメガセルの概要を表3に示す．

表3 CMOS ASICのメガセルの概要

ASICの種類	使用個数				ASIC当たりのメガセル数	RAM/ROMのビット数
	RAM 256×40	RAM 1K×40	RAM 2K×44	ROM 2K×44		
P1	24	8	0	6	38	1,114,112
P2	1	12	0	0	13	501,760
SA	18	12	0	0	30	675,840
SD	0	0	14	0	14	1,261,568
合計	43	32	14	6	95	3,553,280

RAMおよびROMのメガセルは，いずれもそれぞれのRAM/ROMごとにBIST（組み込みセルフ・テスト）機能を持っている．BISTは，組み込みのメガセルが，ダイ・レベルのテストからフィールド診断テストでの使用に至るまで，完全に機能を果たすことを検証するために使用される．

BISTは，11 nsのシステム動作速度でメガセルを実行させることができる．このテスト速度は，メガセルが通常のシステム運転でも作動することを高いレベルで実証することができる．

ROMは，ハードウェアにプリミティブなMISR（Multiple Input Signature Regis-

ter) を用いて，指定されたROM内にある全ビット数の値を収集している．この最終のMISR値は期待値と比較されて，最終的なBISTメモリの有効性チェックを行っている．

BIST資源はシステムの状態を保存し，その後，効率の良い方法で再初期化するためにも用いられている．これにより，非常に高速なシステム起動や再構成動作が可能になっている．

5) 更なるチップ信頼性の向上

設計手法やテスト方法を改善することにより，チップの信頼性が高められている．IDDQと呼ばれる極限の電流測定技術により，極小のリーク電流の検出するを可能とした．リーク電流を検出されないままの状態にしておくと，製品寿命が短くなる．これらの測定方法は，改善されたRAMのカバー領域を含め，チップのすべてのロジック領域をカバーする．バーンインテストも実施されており，製品寿命が下り坂となった頃まで現れない可能性のある障害を事前に明らかにする．チップはすべて長時間にわたって，可能な最高温度と最大電圧でのバーンインテストにてRAMを含む全ロジックの動的作動確認が行われている．

6) 新しく開発されたチップの実装方法

各ダイはそれぞれ面を下に向けて，IBM社によって開発されたC4 (Controlled Collapse Chip Connect) 方式によってセラミック製のパッケージに接着されている．C4プロセスは，ごく短距離の電気接続をも可能にする．この接着と相互接続の方法により，優れた電気的な効果と信頼性を得ている．電源接続はチップの全面で行われており，抵抗性の損失による電圧変動を非常に低いレベルに抑えている．シリコンのダイは，特殊なエポキシに似た合成樹脂によって下側部分を埋められており，はんだ付けの盛り上がり部分を加熱の繰り返しによって発生する機械的な力から保護している (写真5)．

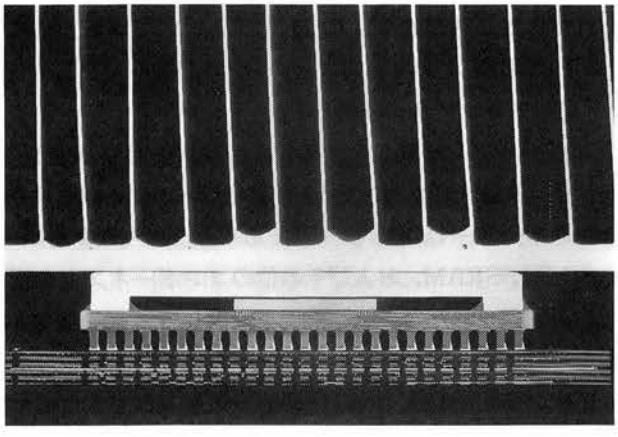


写真5　ASICパッケージの断面図（リッドとヒート・シンクの取り付けられた状態）

パッケージは32.5 mm角で，アルミナ・セラミックからできており，12の金属層を持っている．信号層はすべて電圧層とグラウンド層に挟まれており，高いシグナル品質を保っている．スイッチング・ノイズは設計により，300 mV以下に抑えられている．この値は業界における最良の値である．パッケージの底面にあるピンは耐熱はんだからできており，50ミル間隔の中心上に置かれている．全部で613個のピンがあるが，そのうちの468個が信号用に使用できる．9.95 mmおよび15.5 mmのダイのパッケージ・サイズは同じで，ピンのパターンも同じであり，外見上その相違はない．

パッケージは，耐熱はんだからできているカラム（ピン）によってプリント回路基板に接続されている．はんだカラムの長さは正確に制御されており，標準的な表面実装の技術によってプリント回路基板に接続できるようになっている．リッドは，接着材を用いてパッケージに接着されている．また，熱伝導性の高いグリースによって，ダイとパッケージ・リッド間の熱伝導が効率良く行われるようになっている（写真6）．

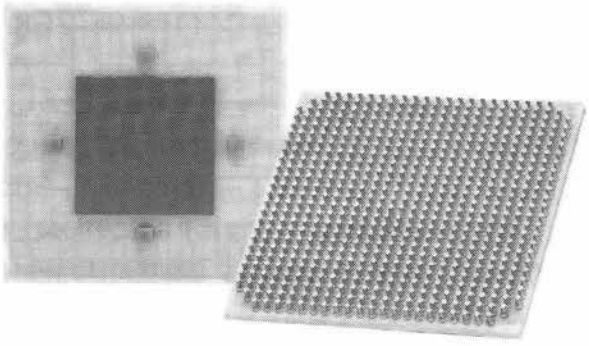


写真6　ASICパッケージ（リッドのない状態）

空冷に用いるヒート・シンクは，スプリング・クリップによってパッケージ・リッドと常時接触している．熱伝導性のグリースが，パッケージ・リッドとヒート・シンクの間にも使われていて，熱が効率良く伝わるようにしている．これによりダイの最大温度は75°C以下に保たれている．

7)　CPMプリント回路アセンブリ（写真7）

CPMではCMOSチップを搭載した2種類のプリント回路基板が用いられており，一つはインストラクション・プロセッサ/ストレージ・コントローラ（IP/SC）ボードであり，他の一つはメモリー(MSU)ボードである．これらの多層ボードは，ボード，CMOSチップ，SIMM，SRAM，およびその他のコンポーネントを相互接続するための，インピーダンス制御された伝送線路環境を提供している．確立された素材やプロセスを使用することにより，高品質と低コストが実現されている．ボードの素材は，長年にわたってプリント配線基板（PWB）製造で使用されてきた標準的な耐熱多機能のエポキシ/グラスで，標準的なFR 4エポキシ/グラス素材よりも簡単にPWBの組み立てが可能で，信頼性も高い．標準的な銅電気めっきのプロセスがメッキされたホールによる相互接続に用いられている．CCGA（セラミック・カラム・グリッド・アレ

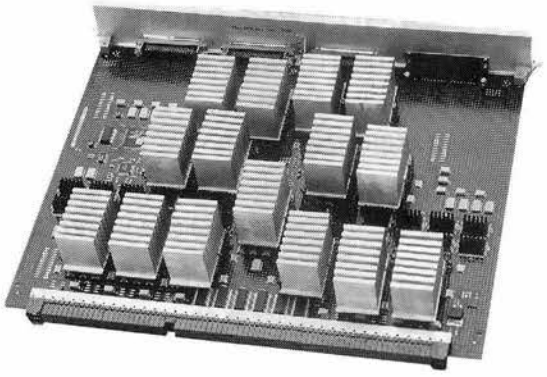


写真7　27.94 cm × 36.58 cm のPCボード上に，四つのプロセッサ，ストレージ・コントローラ，および2次キャッシュを持つITASCA3800インストラクションプロセッサ・ボード

イ）のASIC CMOSパッケージを含むすべてのコンポーネントのはんだ付けが確実に行われるよう，すず・鉛はんだに替わり固有のはんだが用いられている．

両ボードともインナー・バイアがあり，層間の相互接続に0.18 cm当たり2本のルートを持っている．表4に，ボードの物理的なデータを示す．

表4　IP/SCボードとMSUボードの概要

	IP	MSU
幅(cm)	27.94	27.94
高さ(cm)	36.58	36.58
厚さ(cm)	0.254	0.229
信号層の数	10	8
層の総数	22	18
線幅	.0035	.0035
インピーダンス(Ω)	50	50
インナー・バイア	有り	有り
CCGA	採用	採用

2 mmコネクタおよびSIMMコネクタを除き，上記のすべてのボードで0.18 cmのグリッド上にある613個のパッドを持つCCGA ASICを含めて，SMT（表面実装）が用いられている．SMTを積極的に使用し，PWBの一方の面だけにコンポーネントを配置することにより，一回のはんだ付けでアセンブリを可能にし，より信頼性の高いプリント回路基板が確実にできるようにしている．

IP/SCボードには，CCGAのASICと0.064 cmの高精細のピッチの2次キャッシュRAMが載っている．MSUには，CCGAのASICとSIMMメモリ・モジュールが載っている．

I/Oバックパネルに接続される主なプリント回路基板はCSIOPである．このボードはCMOSチップとUNISYS 2200/500プロセッサの技術を用いている．18層のボ

ードにより，インピーダンス制御された回路が，CMOS チップとその他のコンポーネントを相互接続している．ボードの素材は，標準的な耐熱多機能のエポキシ/グラスである．層間の相互接続を行うベリード・バイアには，0.11 cm 幅の信号線が用いられている．すず・鉛はんだに替わり専用のはんだを用いることにより，すべてのコンポーネントのはんだ付けが確実に行われるようにしている．高精細ピッチ（0.064 cm）の表面実装および 0.18 cm 間隔のちどりピン・グリッド・アレイの CMOS ASIC の構成要素が使用されている．

8) 新しい高密度のメモリ・パッケージング（写真 8）

主記憶総理（MSU）ボードは，ITASCA 3800 システムの 3 次レベルの記憶装置を担っている．各ボードには，それぞれ四つの制御用 CMOS ASIC と最大 40 枚の SIMM ボードが搭載される．

1 枚のメモリ・カードは，64 メガ語（256 メガバイト）から 256 メガ語（1 ギガバイト）まで記憶容量を拡張できる．64 メガ語（256 メガバイト）のメモリ拡張機能は，10 個のプラグ接続可能な SIMM を使用している．SIMM はそれぞれ，18 個の表面実装された 4 メガバイト×4（16 メガバイト）の DRAM デバイスを搭載している．

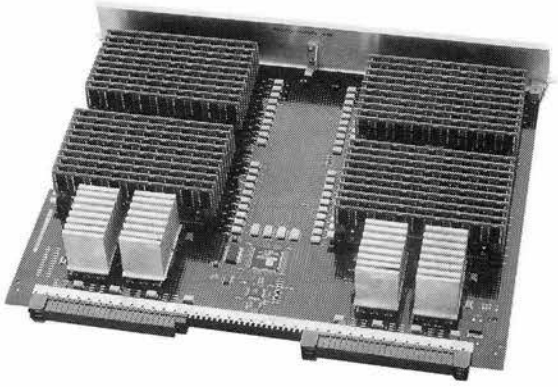


写真 8　1 ギガバイトの記憶容量を持つ 2200/3800 主記憶ボード

9) 電源システム

電源システムは，信頼性の高いスイッチング方式の電源ファミリを使用している．電源システムは非常に耐障害性に富み，システムの運用に影響を与えることのない設計となっている．この障害許容機能は冗長電源によって実現されている．一つの電源がダウンすると，並列に接続されているモジュールが負荷を処理する．電源制御部はイベントを報告するが，システムは中断されることなく動作し続ける．障害を起こしたユニットは，後で都合の良いときにいつでも交換できる．さらに，電源は「電源を入れた状態」で着脱（活線挿抜）できるように設計されているので，この交換はシステムの動作を中断させることなく行える．

電源システムは，世界中の広範囲にわたる電圧や周波数(公称 200～240 V，50 Hz ま

たは60 Hz)で動作するように設計されている．障害許容機能を強化するために，ユニシスでは，絶縁変圧器，AVR，および，UPS製品を含む幅広い製品を提供している．

電源制御にはマイクロプロセッサが使用されており，SCF（システム制御装置）と関連して作動し，電源モジュールのステータスの制御，監視，および電圧，エアーフロー，および温度についてのステータス報告を行う．

3. ま と め

本稿では，ITASCA 3800シリーズの新しいハードウェア技術の主な特長について説明した．あらゆるレベルにおいてCMOSチップ技術および最新の実装技術を使用することにより，今日の競争の激しい市場において求められている，低コスト，高性能，高可用性，および，システム規模の変更等を可能としている．設計技術，設計ツール，および開発プロセスにおける多くの他の技術革新については，本稿では言及していないが，ITASCA 3800システムの先進的技術の採用についての方法論を示した．

エンタープライズ・サーバの現状と展望

本討論会は，メインフレームとオープンシステムの特質について明らかにし，それぞれの適用の方向性を探ってみようとしたものである．オープンシステム採用の狙い，オープンシステムへの期待，利用形態，運用管理の課題，バッチ処理時の課題，障害対応・プロダクト間の整合性，開発手法，開発支援ツールの課題，などについてオープンシステムの特質を語り合った後，今後のメインフレームのあり方について述べている．

出席者：
遠藤和弥　(オープン技術部)
馬場功二　(オープン技術部)
神谷是公　(システムプロダクト部)
今江　泰　(エンタープライズサーバ企画推進部)
田辺英夫　(オープン企画推進部)
司　会：
松倉　司　(システムプロダクト部)

オープンシステムの狙い

司会　この討論会では，メインフレームとオープンシステムについて，それぞれの特長とどの様な使い方が適切なのかを語っていただきたいと思います．

数年前よりメインフレームの役割は終わったとの論調が出てきています．オープンシステムと比較してメインフレームに技術的な弱みがあるために，このように言われるのだとは思い難いのです．技術的な観点からみればオープンシステムは，まだまだメインフレームの持っているレベルに到達していない部分が多いのではないでしょうか．それにもかかわらずオープンシステムに移るべきだと言われるのはどのような理由によるものなのでしょうか．

まず最初にユーザはオープンシステムに何を求めているのかを語ってみていただきたいのですが．

今江　日本と米国の事情は同じではないと思います．日本では明らかに不況期に入った時にメインフレームを離れ，コスト削減のためにオープンシステムを採用し，ダウンサイジングを行うべきと言われ始めたような気がします．しかし，市場動向を見ていると，メインフレームは簡単にはなくならないし，オープンシステムと共存共栄で使われていくケースがまだまだ多いと思います．

米国でのアンケート調査の結果では，ユーザはシステムを高度化し企業力を強化する事がプライオリティ第一で，コスト削減がメインの理由ではないのです．96年2月のガートナの報告会で示された調査レポートでは，クライアント/サーバ・システムの開発プロジェクトのほとんどは，今までのシステムと比べて，期間もコストも増えている．それでも，7割のユーザは，プロジェクトは成功したと言っています．機会があれば，次もクライアント/サーバで作っていくと9割以上が言っている．

少し違う話ですが，米国でも一番最初は不況がもとでダウンサイジングに向かったと思う．でも，通産省の資料によると，米国は不況になった時に情報に関する投資だけは減らしていない事が示されています．日本はそれを減らしてしまった．それが今や情報処理分野だ

けでなく国力の差になっている．どうも残念な状況になってしまっていると思うのです．

遠藤　私もまったく同感で，日本は不景気とオープンの波がぶつかった．そこで，ダウンサイジング＝コスト・ダウンサイジングを狙った．日本ではマイクロソフト社のWindowsNT*1が米国以上に受け入れられているのはコスト・ダウンサイジングの先兵だと市場が期待したからだと思う．UNIX*2サーバで実現するのか，WindowsNTサーバで実現するのかといったら，WindowsNTサーバの方がずっと安い．そこにユーザが魅力を感じた．どこまでできるかということよりも，ひたすら安さに目が奪われたと思う．

では，今どうかというと，確かにできるところもある．でも，本当にどこまでできるのか．もう一つは，日本のユーザの側でも，コスト・ダウンサイジングにならないなというところが見えてきた．オープンで，クライアント/サーバで，分散でシステムを再構築した時に，メインフレームの集中と比べたらどうも安くならない．しかも，運用コストとか人件費がばかにならない．また，ファット・クライアント・シンドロームと言われているような，大きなクライアントを大量に配置するというシステムの作り方も，高コストをもたらす要因になっています．

そこで，今一度，分散すべきものと分散してはいけないもの，技術的にはできるかもしれないが運用管理を含めて分散してはいけないもの，それを見直そう，データの配置をもう一度見直そう，それによってシステム全体のコストを見直そう，という意見が出てきています．日本ではコスト・ダウンサイジングを狙い，ハードウェアのコスト，ソフトウェアのコストでは狙いを達成したが，運用コストという面では，コスト・ダウンサイジングにならなかった．

情報系のエンドユーザ・コンピューティングで，ある部門だけちょっとやってみたというところはまだ良しとしても，少なくともある事業所単位，ないしは企社の共通業務を現在のクライアント/サーバシステムで分散，ダウンサイジングしようとしたところは，狙いどおりのコスト・ダウンサイジングにはならなかったというケースはあると思います．

コスト・ダウンサイジング以外の狙い

司会　オープンに期待したコスト・ダウンサイジングの実現は期待通りにはならなかったという状況ですが，今ユーザはオープンシステムにコスト・ダウンサイジング以外の何を期待しているのでしょうか．

遠藤　それは一般的に言われている特定のベンダーに依存しない，いろいろなベンダーのものを自由に選択できるという事がオープンに対する期待そのものだと思います．

今ユーザと話していると，極端な話はハードウェア，ソフトウェア，サービスすべてを含めてできれば1社のものがいい．それができないなら，例えばハードウェアはコンパックのPC，OSはマイクロソフト，データベースはORACLEのデータベース，それぞれ1社にしたいと言う．データベースをinformix，SYBASE，ORACLE*3と三つを使いたくはない．ハードウェアも，DOS/V*4マシンといってもみんな違う．DOS/Vだというだけで，コンパックだ，DECだ，エイサーだ，どこから調達してもいいということにはならない．ハードウェアもどこかベンダーを一つに決める．

おかしな話で，メインフレームの時と何が違っているんでしょう．オープンだ，オープンだ，選択の幅があってどこから調達してもいいはずだと言っているけれども，今はどうも違ってきている．以前に戻ってきている．以前と同じような考えには戻らないかもしれないけれども，オープンに対する期待というのは，どこから選んでもいいというふうにはならないという事に，ユーザが少しずつ気がつきはじめた．

神谷　私もあるオープンのユーザの実験システムに関するサーベイレポートを読んだのですが，確かに選択肢はたくさんある．しかし，組織業務の中で2，3選択したら，それを

取り替えるというのはほとんど不可能だというのが,実際に使っている人の経験らしいのです.

そもそもオープンというのは何を狙ったのか.一般的なオープンの狙いということでは,私は二つ程あると思っています.一つは,従来からメインフレームが行ってきた基幹業務をベースにした情報活用に対して,個人がどれだけ生産性を上げていくか,組織業務の中および個人のレベルで情報活用力を高めていくという事が,オープンの大きな狙いの一つだと思います.PCを大量に配置することによって,現実のニーズになりつつあり,そこでいま一生懸命実践している段階ではないかと思います.

もう一つの狙いは,グローバル・ネットワーキング,ネットワーク・コンピューティングと呼ばれていますが,意思決定をいろいろなレベルでする時に,情報を集中し,それに基づいて発見的な手法を使いながら一つのデシジョンを下していく,あるいは一つの業務処理をするうえでも,ユーザのニーズをどう的確につかまえ,どれだけ速く処理するかという事です.例えば受発注のトランザクションをインターネットを介して処理していくシステムはそういうところを狙っているわけです.

オープンが本来狙っているのは,個人個人が情報を活用して生産性を上げていくという点,もう一つはグローバル・ネットワーキングを通して,競合他社のいろいろな情報を見つめながら意思決定していく,あるいは消費者と迅速にうまく結びついてビジネスをやっていく,というところではないでしょうか.従来の基幹業務系,メインフレームだけで行っていた10年前から見ると,その二つの点が大きく狙っているところであるという感じがします.

馬場　かつては基幹系があって,情報系ということだったのが,今は情報系という大きなものがあって,基幹系ということになっている.

それに対する道具としてうまくフィットしそうだったのがオープンシステムで,オープンシステムはそこに対する期待が一番大きかったのではないでしょうか.

今それに輪をかけているのがインターネットであり,さらにもう一段,今までのクライアント/サーバよりも値段が安くなるのではないかと言われているのがイントラネットということになると思います.

オープンシステムの利用形態

司会　オープンシステムに対するユーザの期待するところについてまとめていただきましたが,オープンシステムの使われかたの現状はどうでしょうか.適用の形態であるクライアント/サーバシステムについては第1次段階と第2次段階という言い方があって,従来は第1次で,これからが第2次だと言われています.第1次は基本的にはUNIXサーバで,データベースをのせて,PCを接続するというのが一番多かったかと思います.日本の第1次というのは,どういう分野で主に使われていたのでしょう.先程言われた情報活用に近い分野でしょうか.

馬場　情報と基幹の中間的な業務ではないでしょうか.少なくともUNIXの第1次の業務というのは,現在のEUCのところまではいきませんでした.基幹もあまりやらなかった.その意味では中間のところだったのです.さらに処理形態は決してクライアント/サーバではありません.ホストのオープン化,ホストのUNIX化です.当初のシステムは,端末はほとんどVT端末ですから,そういう意味ではホスト—端末の世界とUNIX—PCの世界は構成的にはほとんど同じです.クライアント/サーバで実現した例はあまりないのです.そのあとPC,クライアント側にアプリケーションとGUIをのせた本格的なクライアント/サーバになってきたと思っています.

遠藤　クライアント/サーバというのは本当にいいのか.確かに特性はありますが,みんなが間違ってしまったのは,これで全部できるのではないかと思ったところです.SQLを通

してやりとりするクライアント/サーバに向いているところと向いていないところがあるのですが，クライアント/サーバはこれだと一つに決めこんで，それを全部に適用しようとしたから間違ったのではないかと思うのです．

最近はそれに対する反省として，ファット・クライアント・シンドロームではいけない，だから3階層だと言われている．たぶんどちらかという話ではなく，2階層もあれば，3階層もあれば，n階層もあれば，いろいろなクライアント/サーバがあっていいと思うのです．その意味では，まだまだ発展途上で，いままでのクライアント/サーバは2階層クライアント/サーバという一つの処理形態でしかないのです．

運用管理の課題

司会　オープンシステムには，従来のメインフレームの基幹系がうまく取り込めないという指摘もありましたが，まだこのへんが問題だというところはどんなところでしょうか．運用管理が大変だということを聞きますが，オープンシステムの運用管理では具体的にどういう課題，限界があるのでしょうか．

遠藤　今クライアントのPCを1人1台とする事を目指していますが，1人1台のPCが配られた時にどうなるのかを本当にわかっているユーザは非常に少ないと思うのです．いまのPCはテレビとか電話みたいなものではなくて，非常に手間がかかる．買ってきて，線をつないで電気を入れれば使えるというふうにはならないわけです．誰かがディスクのパーティショニングをしたり，OSのロードをしたり，ソフトウェアの落とし込みをして，その上でさらに環境設定をする．それをLANにつないで，メールを使ったりグループウェアをやろうとすると，ソフトウェアごとに環境設定を行わなければならない．しかも，それらの整合性をとった環境設定が必要になってくる．非常に手間がかるのです．

昔のホスト―端末の世界の専用機では，単機能ですから，何も設定が要りません．ソフトウェアが落とし込んであって，電源を入れたらあるメニューが出てきて，キーボードの操作も同じということでした．こういう世界であれば，教育もあるところで集合教育したら終わりということで，コストもあまりかからなかった．

今は一人ひとりが違う操作環境を持つわけです．コンピュータのコの字も知らない人たちが机の上にコンピュータをどんと渡されても，とても自分ではできません．誰かがサポートする，ないしは代わりにやってあげるということが必要になってくる．したがって，大変なコストがかかってくるというのが現状だと思います．

これは小規模の時はそれほど目立たないのですが，規模が大きくなればなるほど，大きな問題になってきます．この問題が一番大きいのかと思います．

神谷　それはサーバにも同じことが言えます．クライアントが1人1台になって，そこでちょっと古いソフトウェアを動かしたというと，純粋にパーソナルに使っている限りは大して問題にならないけれども，互いの部署が関係して，その一つとして使っている時はたぶん許されない．何百台から何千，何万台のオーダーになると，そういうところをどうやって管理していくのか．さらに，そうなってくると当然サーバもいくつかに分散されて，データの整合性，あるいはシステムそのものの運用管理の面で，管理しなければいけないサーバもあちこちに出てきてしまい，問題が大きくなる．

今江　サーバを入れる計画を持つユーザの例で，置き場所は1か所にしたいという話を時々聞きます．あるユーザの例で，PCのNTサーバが10台，クライアントが300台のシステムを検討中ですが，サーバは1か所に置きたいと言っています．別のお客さんからも聞いたことがあるので，管理する立場からいくと集中で運用管理したいという要求が強くあるのではないかと思います．

遠藤　SPARC[*5] 2000が8台，同じところに並んでいるユーザの例もあります．サーバの

運用管理の問題は，クライアントよりは少しは楽なんです．個人ごとの設定はなく，共通環境で，しかも個人からその環境を変えるということは普通はない．サーバの管理というのはメインフレームが分散したのと同じと考えればいいと思います．だから，比較的やりやすい．

ただ，データベースを持った場合に，そのデータのバックアップとかプログラムのバックアップを誰がどのように行うのかは問題になっています．全社の情報システム部門の担当とすると，そこに人を置いて行うのか，それとも無人で自動的にバックアップをとるようにするかが問題になります．難しいのは，部門管理のサーバとした時で，データ管理者は誰かを決めるところから始めなければなりません．

運用管理というのは非常に難しい問題で，誰の管理物か，誰の責任保管のものかというところから決めなければならない．運用管理と一口で言えない難しい問題をはらむことになる．いままで使ってきているサーバで，情報システム部門が管理という場合は，いままでの延長線上でいけるのですが，EUCとか情報系といって，とくにEUCで部門ごとにサーバを置いて，その部門でサーバを管理しなさいというと難しい話になってきます．たぶんその部門に小さな情報システム部門がなければいけないという話になってくると思うのです．

田辺　部門でもある程度力をつけていかないと，結局は与えられた情報だけを使って仕事をするだけという事になってしまうと思うのです．だから，会社が大きい組織で，種々の有効な情報を与えられるような組織として情報システム部門があるならば，一括管理する部門として情報システム部門があるのも良い方法だと思います．ところが，ビジネスにおいてはボトムアップ情報のようなものも重要です．だから，与えられた情報を処理するだけでなく自分で情報をクリエートする事ができるようにならなければなりません．そのためにも自部門で情報管理が出来るようにならなければいけないと思います．

今はどうしても運用管理は情報システム部門に頼みたいという発想が強いのではないかと思います．なかなか次のステップに行けない．

遠藤　本当は現場にある生のデータが一番いいということと，日々の活動から自分が情報を発信していくということが大事なのです．今まではそういうことがうまくできる道具もなかったと思うのです．今はLotus Notesやメールで，それぞれの個人が情報を発信できるようになってきたわけです．

田辺　部品が結構出てきて，環境はある程度揃ってきたという感じはします．

今江　個人または部門で情報発信をできるようにするという意味では，部門ごとにNotesなりWebのサーバを持っているという方向に行かざるをえませんね．そこの管理は情報システム部門ではなく，部門にそういう役割を持った人を置かざるをえない方向でしょうか．

遠藤　そう思います．われわれもボランティアで始まったんですね．PCが2，3台あった．5台になった．どこかにちょっと大きなPCを置いて，Netware*6を置いて，プリントサーバとファイルサーバにしましょう．そこまで来ますとボランティアでは済まなくなってきます．百数十名もの規模となると，そのネットワーク管理，サーバ管理をボランティアでやろうと思ってもできません．われわれの中でも情報システム部門が要るんです．少なくともこれぐらいの規模では，それを考えている人，それを見ている人がファンクションとして要るんです．これだけの構成，管理をするだけでも大変です．

例えば引っ越しがある．では，どういう構成にしようか．将来的にどういうネットワークに持っていこうか．これは人間が考えなければいけない．今のトラフィックがどうなっていて，これからの使い方はこうなるはずだから，こういう構成にしていこう．これは誰かが考えなければいけない．その意味で，この中の情報システム部門として誰かが要る．

今江　そこにかかるコストは企業が吸収しなければいけないコストであって，コストがかかるからやめてしまおうというものではありません．米国では，企業としては必要なコストであると認めているのです．

最近思うのは，文化風土，環境がすごく大事だと思うんです．そういうものに対してお金をかけなければいけない時代になってきている，かけて当然だというのがないと，少なければ少ない方がいいという話になってしまう．でも，セキュリティの問題も，道具があればいい話ではなくて，文化なんですね．ソフトウェアに対してお金を払う．これも一つの文化ですね．そういう文化風土がないと，こういう世界は価値観を見い出せないというか，うまく回らない．われわれも日本の文化風土をそういう方向に変えていかなければならないと思うんです．

バッチ処理等について

司会　オープンシステムの運用管理の課題をお話していただきましたが，情報システム部門のあり方についてのお話にまで広がったかと思います．これは重要な課題の一つである事は確かですがひとまずおきまして，もう一つオープンシステムの運用管理の問題として，メインフレームで行っていた業務，例えば夜間のバッチ処理をやろうとすると，いまのオープンシステムではやりにくいという話を聞くことがあります．それはどういう機能が十分ないからやりにくいのでしょうか．

遠藤　勘定系と情報系の中間のところをUNIXでやった時は，やはりバッチ処理業務はありました．というのは，基幹系との連携があって，なんらかの集約データを作成して，それを渡さなければいけない．バッチのスケジューリング機能とか，バッチ処理はどうしても必要でした．例えば役所への提出資料というと，帳票がたくさんあるわけです．こういう帳票はどうしてもバッチ処理で作らざるをえない．月報，季報，年報みたいなものを作って出さなければいけない．こういうレポーティング処理は，データを集約して，バッチとして動かす事になります．

神谷　それはUNIXで処理するとコストダウンになるということから，従来メインフレームで処理していた業務をどんどんやってみようという背景なんでしょうね．その時には，処理ではI/Oのスピードはどうだとか，大量の印刷物が所定の時間内に印刷し終わるのかというところがたぶん問題になると思うんです．そこの場面で適用のしかたを間違ったのではないでしょうか．

司会　そういう局面にはオープンシステムは時期早尚だったという意味ですか．

神谷　ええ．

遠藤　でも，UNIXはスケジューリング機能や編集機能も結構充実はしてきたんですね．メインフレームほどではないんですが，かなりメインフレームに近づいている．今PCサーバはUNIXをさらに追いかけている．

今江　バッチといっても処理量や処理内容が影響するのではないかという感じがします．というのは，一つの事例は某金融機関ですが，今までメインフレームでやっていたシステムをオープンシステムに移行することを検討した．トランザクション系はUNIXベースで結構いけそうだという見通しがついた．ところが，バッチだけはどうしてもできない，それをどうしようかということが問題になったようです．

また，別の話で，ある流通関係の企業が今までPOSから集めたデータを1日に何回かに分けて某社のメインフレームで処理していたが，2時間以上かかり発注時間に間に合わなくなってきた．そこでUNIXで処理してみたら，何も手を加えないでも20分に減った．さらに工夫してやってみたら，もっと減ったという話を聞いたんです．

この二つの例は，片方ではメインフレームである時間にこなせなかったものがUNIXで処理できたと言っていて，一方ではとても無理だと言っている．バッチ処理の内容とか，データの構造とかが絡むのではないでしょうか．一概には言えないとは思います．

遠藤　それはあると思います．本当はメインフレームはデータウェアハウスには一番いい

と思うんです．なぜかというと，大量のマスストレージを構成に含められますし，大量のデータに同時にアクセスする力を持っています．UNIX も大量のデータを構成に含める事はできるのですが，大量のデータに同時にアクセスできるかというと否なんです．そこは違います．そういった意味では特性はあると思います．ただ，CPU を何枚も入れて速くしていく，SPARC の速い石を並べる，HP の速い石を並べるということはできると思います．だから，一概にこれはできて，これはできないという話ではなく，処理の中身，特性だと思うんです．

田辺　例えばテープの I/O は，UNIX と比べるとメインフレームのほうが段違いに速いんです．だから，メインフレームがいいか，UNIX がいいか，というのは，ケース・バイ・ケースで分けていかないといけない．ただ値段が安いだけというのではきついのではないかという気がします．OLTP の場合，プログラムがメモリに張り付いているケースが多いでしょうから，単に CPU 能力だけで言ったらどちらの方がいいと言う話になることがあるかもしれません．

今江　I/O 能力から見てもメインフレームの方がまだ強いですね．

遠藤　デュアルアクセス一つとっても，そうですね．

障害対応・プロダクト間の整合性

司会　オープンシステムの問題点としてバッチ処理を例にしてスケジューリング機能・I/O 機能の話をしていただきましたが，対象業務の特性を十分に認識適用すべきということになるかと思います．次に，オープンシステムの障害対応，プロダクト間の整合性というところでの現状・課題についてはいかがでしょうか．

馬場　障害対応からすると，UNIX もメインフレームの機能をだんだん備えてきていますし，PC もそれを追いかけているという状況だと思います．ただ障害が起きた時の対応というのは，メインフレームは一つのメーカーが全部の責任を持っていますので，対応力も集中できるというか，障害の押さえ方，手法，経験を十分備えていると言えます．一方，オープンシステムになると，集中していなくて分散している．しかも線でつながっているとなると，どこが問題の箇所かを詰めるまでにかなりの時間を要する事になります．もう一つは，それぞれ担当している会社，製品のメーカーが全部違うということも，時間のかかる大きな要素だと思います．

田辺　先ほど**遠藤**さんが言われたんですが，なるべくだったら１社，もしくはコーディネーターが１社，いろいろな意味で対応能力のある会社がいいという話になってくると思います．

馬場　整合性という観点から言いますと，単体の製品としては問題がない．しかし，組み合わせて使用することにより，または数が多くなることによって，予想しない，経験しない問題が現れてくる．これもメインフレームだとかなり歴史もありますし，いろいろな点で限界値もある程度わかっていますし，技術者もかなりいる．しかし，オープンシステムの場合はまだ経験者が少ない．したがって，初めて経験する問題にぶつかるケースが多いのではないか．そういう点で，障害，整合性というところは，非常に関係があるのではないかと思います．

司会　それはオープンシステムのプロダクトをよく知った技術者が多くなれば解決していくと単純に言えるのでしょうか．

遠藤　オープン製品の特性として，なぜオープンの製品は安いかというと，ハードウェアもソフトウェアも障害のことはあまり考えていないわけです．非常に割り切りの多い世界です．一つ一つが割り切られていますから，使う側としてはそういう世界だと思って使わないといけない．

例えばメインフレームでやっていたトランザクション処理みたいに，送信したらどこかで

誰かがリカバリーしてくれて，メッセージの保証をしてくれるという世界を実現しようと思ったら非常に大変です．オープンの世界でそれをやろうと思ったら，TP モニタを入れて，しっかり作り込んでやっていかなければいけない．

ハードウェアとソフトウェアの接点である OS 一つとっても，メインフレームはアプリケーションまで一緒でしたが，UNIX の場合はアプリケーションは別になりますがまだハードベンダーと OS ベンダーが一緒です．PC になるとハードウェアと OS も分かれてしまう．そうすると，その間の問題になった時に，障害原因の切り分けが大変です．メインフレームの場合のように切り分けということを考えていてはいけない．何かあったら，パーソナルの世界ですから，やり直してみる．再現しなければそれでいい．こういう割り切りをしていかないといけない．われわれが基幹系でやってきた価値観ではない価値観で使っていくものではないかと思います．では，それで使えないのかというとそうではない．例えば今のメールがそうだと思いますが，メールが届かない，どこでどうなったんだと原因をとことん追求するかというと，それよりも再送する．こういう文化です．それによってメールは使えないとは誰も言っていません．ですから，メインフレームのコンピュータとパソコンを中心とするオープンのコンピュータは，まったく別物だと考えた方がいいのではないかと思います．

司会　そういう話を聞くと使い分けの話になってしまいますが，従来の基幹系業務(入出金処理，発注処理というような業務)には，オープンシステムは使えない，使う際には相当心して取り組む事が必要という話になるのでしょうか．

馬場　今の時点ではそうですが，これから技術がどういうふうに変わってくるのか，補うものが出てくるのかどうかはわかりませんが，ミッション・クリティカルな業務をメインフレームに変わってオープンシステムがまだ完全には行っていけない一つの要素だと思います．

とくに銀行業務のような，一銭たりとも狂ってはいけない，誰が何をやったかが一目瞭然でわからなければいけないという業務特性を持つものについては，オープンシステムが出来ないとか出来るではなく，作られ方，考え方といったところも，現時点では適していないというところがあるのではないかと思います．

遠藤　一つには，数字で 1 ビットも落としてはいけないという世界があります．そういう世界とインターネットでのマルチメディアみたいな世界があります．この世界は 1 ビット落ちようが，2 ビット落ちようが関係ない世界です．1 ビットも落とさないという世界と，イメージとか動画みたいにひたすら大量に送りたいという二つの世界があり，たぶん技術が違うと思います．

ガートナなんかも，部門サーバとしての NT はあるかもしれないけれども，誰も MVS を NT に書き換えようとは思っていない．MVS のアプリケーションを NT に乗せるということはないだろうと言っています．そういう棲み分けになっていくのではないかと私も思います．

開発手法

司会　メインフレームとオープンシステムの比較でもう一つ開発手法の違いが大きいと言われています．オープンシステムのソフトウェア開発手法や発想法はメインフレームと比べてどう違うのでしょうか，それとも同じなのでしょうか．また，生産性はメインフレームと比べて高いのでしょうか，同じなのでしょうか．

田辺　ソフトウェア開発手法は時代とともに進歩していると思います．ですから，オープンだから，メインフレームだから，という話ではなく，良い開発手法を使えばいいだけです．

遠藤　ともするとメインフレームはウォーターフォールで，オープンはプロトタイピングの，スパイラルのと，思われていますが，私はそうではないと思います．メインフレームの

大規模開発の時代の開発手法としては，ウォーターフォールはすごくフィットしたと思います．もしあの時代に今みたいなプロトタイピングだ，スパイラルだとやっていたら，たぶんうまくいかなかったと思います．あれはあれで一つのやり方で，正解だったと思う．

では今オープンの世界で，ウォーターフォールは間違いですかというと，私は全然そうは思わない．ウォーターフォールの世界のやり方，ある工程をきちっと決めて，次の工程に進んでいくというところは，必要な局面がいっぱいあるわけです．それだけでいいかというと，それだけではないよというところがあるので，プロトタイピングとかスパイラル型を取り入れていかなければいけない．

一方，ジェームス・マーチンが言っているRADの世界を日本でやろうと思うと，現実は非常に難しい．RADのチームは，少数精鋭のスペシャリスト集団を短期間で結成して，短期間で開発して，そのチームを崩さずにどんどんやっていこうと言っています．日本でそれをやろうと思うと非常に難しい．例えば会社の中一つを取っても難しい．われわれも経験していますが，ユーザがあって，ユーザの系列の会社があって，その下にソフトウェアハウスがあって，これだけでチームを作りましょうといっても，まったく別法人で，それぞれ人の育成を考えているとか，難しい土壌がいっぱいあってうまくいかない．

ただ，考え方はいいと思います．ああいうやり方をして，そこにコンピュータテクノロジーの最新のCASEツールを使っていくのはいいと思うので，どうやったら日本型のRADがやれるのか．日本型のスパイラル型がやれるか．ウォーターフォールとスパイラル型と日本型RADを組み合わせたような開発スタイルを作らなければいけないと思います．

メインフレームの時もまったく同じだったと思いますが，ものさしは一つでは駄目なんです．当社はRSDMということで開発標準を作った．では，これ一つで大規模から小規模までいきますかというと難しいと思います．オープンの世界は，少なくとも3種類は要ると思います．比較的大きなもの，中くらいのもの，1か月～2か月でやる小さなもの，その三つくらいのものさしというんですか，方法論と手順と道具，そしてそれができる人(人というのはスキルを持った指導できる人という意味です）が必要と思います．

オープンの世界での開発は，いま残念ながらうまくいっていません．でも，そういったものを作ってうまくやっていかなければいけないと思います．今うまくいっていない開発の一番の問題は，オープンの世界という新たな環境で仕事をするわけですが，それに対する突っ込みが足りない．ハードウェアも，ソフトウェアも，ミドルソフトウェアもほとんど他社の世界で作るわけですが，それに対して突っ込みが足りないと思います．ユーザのシステムを作る時に，これはよそものだから知りませんと言っていてはうまくいきません．

プロジェクトマネジメントも同じだと思います．また私はインフラです，私はアプリケーション担当です，私は上流工程の専門家です，私は下流工程の専門家ですなどと言っていては駄目なんです．貪欲になんでもやっていかないといけない．大きいメインフレームをやってきて，分業体制に慣れ親しんでしまって，それに浸っているために，なかなか分業体制から脱却できない．いまは一気通貫で，一人で最初から最後まで全部やらなければならない．一人で，上流工程から下流工程まで，基本ソフトウェアからアプリケーションまでという世界だと思いますが，こういうマルチ人間というかマルチスキルというか，そこが今ネックではないかと思います．

司会　それは間に入るプロダクトとして自前のものだけでなくなっている以上，いろいろな会社のものを寄せ集めて並べなければいけない．多くのプロダクトの機能・使用法・制約等をすべて確認した上でないと，上流から下流まで通しで理解してシステム構築を行うことが難しいということでしょうか．そこをどうやって解決していくかというのが課題になってきましたね．

遠藤　そこでわれわれが考えたのがセットです．毎回別の組合せで，毎回初体験ではどう

やってもうまくいかないだろう．ですから，好き勝手を言えばたくさん出てくるけれども，セットを決めて，そのセットで構築するようにしましょう．そのセットでの要員の育成をしましょう．セットでの売り方を決めましょう．そこで，オープン・ソリューション・フレームワークというものを作っていったわけです．

これは一つのガイドで，当然ガイドからはずれてもいいわけですが，その場合はガイドからはずれたなりの対応が必要になる．少なくともガイドに乗っている限りは，こういうふうにやればいいんですよと持っていきたい．そのフレームワークに沿ってある一部を選んだ場合にも，完全・確実・所定コストで構築できることをユーザに対して保証するわけです．

開発支援ツールの課題

司会　開発環境という意味では，いかにメインフレーム用のアプリケーション開発作業といっても，もうオープンシステムを活用した環境に移っていると思いますが，いかがでしょうか．いわゆるGUIでの開発支援ツールでの課題等はあるのでしょうか．

馬場　環境というのか，どちらかというとツール的なものですが，逆にツールがゆえに見た目のGUIにだまされてしまうというか，そこのところだけで処理してしまうために，表面だけで作り上げてしまうというのが，今の大きな落とし穴ではないかと思います．先程から話が出ているように，全体を考えて，そのソフトウェアを通して向こうが見える形で開発を進めないと，できあがった時には表はきれいだけれども中身はぐちゃぐちゃ，できあがったものは問題山積という形になってしまう恐れがあります．

遠藤　選択肢はいくつかあると思います．サーバ上のアプリケーションはサーバ上の開発ツールを使って作る．ないしはそういったものを使わずに，サーバ上でCでごりごり作る．最近はCOBOLでごりごり作るというのもあります．そういう場合にクライアント側はどうするのかというと，クライアントだけはGUIツールでやるという場合もありますし，クライアント側もVBとかCでごりごり作るというやり方もある．

もう一つは，第2世代CASEツールと言われているForteで4GL，当社の製品でいえばLINCみたいなものを使う方法です．LINCが内部的にクライアントとサーバにうまく処理を分割して，ジェネレートして動かしてくれる．それがForteだと思えばいいわけです．それは開発する人は4GLを一つ覚えればいい．それでGUIも作れるし，サーバ側も作れる．十数個のコマンドしかなくて，簡単に作れてしまう．

ただ，簡単に作れるというのはコーディングの問題ではなく，オブジェクト指向のツールですから，どういうふうにオブジェクトを設計するかというところが一番問題です．そこのオブジェクト指向の技術者，分析から始まってオブジェクトの設計をする技術者が，これからの大きな課題です．

ツールという意味で見ると，オープンのツールはよくできています．業務分析から始まって，DOAのアプローチをどんどん進められ，最終的にはそれがSQL Windowsとか，PowerBuilder*7とか，そういうツールにリポジトリーを通してつながっていく．上流，中流，下流と三者みんな違うという場合でも，リポジトリーでつながるという非常にすばらしい世界です．メインフレームではできなかったことが，オープンでは一気にできてしまったということです．

もう一つは，上流工程のツールがやっとものになってきた．業務分析のところとビジネスダイヤグラムをツールを使って書けるようになってきた．最初に全体的な絵を描いて，その一つをクリックするとその裏側が広がっていく．その間の線をクリックすると，その属性が出てくる．そういうGUIのよさを生かした上流工程の分析整理ツールが出てきて，これがものになってきた．これからの上流工程は，これらの道具をいかにうまく使いこなすかということだと思います．

オープンの世界になっていろいろな道具が出てきて，結果としてどうなったかというと，開発工程が非常に少なくなったんです．いろいろなものを持ってきて組み合わせる事ができるようになった．これからもっと部品を持つようになると簡単にできてしまう．

何が大事かというと上流工程です．下流工程をいくら生産性を上げてももう駄目なんです．1か月，2か月，3か月でシステム開発を終わらせるためには，上流工程をいかに短期間でやるかというところがポイントになってきています．業務分析から始まって，データ設計，データモデリング，そこから具体的に物理設計に落としてシステム構築をしていく，ここをいかに短期間でやるかというのがポイントになってきています．ですから，これからわれわれの技術を上流のほうに向けなければいけないということになります．

神谷　そこは対人間の要素が多いところでもありますね．CASEツールもツールであって，使う人間がうまく使わないと効果を発揮しないと思います．別に否定しているわけではなくて，そういう環境が出てきたというのはいいことだと思います．今までわれわれが紙と鉛筆でやってきたところに道具が出てきた．ある意味では身構えてやらなければできないところ，逆にあまり簡単だと**馬場**さんが言ったように見栄えだけよくてものがないということにもなりかねない．そんなに身構えないでもできるようになってきたということですね．

馬場　GUIで図がぽんと出てきて，いまの構造が一目瞭然で見える．今まではそういうものを描くことにものすごく時間がかかって，できあがったものをじっくり見る時間がなくて，メインフレームでは見落としとか図に表せなかった部分で後で問題が出てきた．今度は逆にそういうものをしっかり見ることができる．その意味で，便利なツールができたことによって，全体が短くなってきているという感じだと思います．

遠藤　当社のオープン・ソリューション・フレームワークの中に，開発のツールと手順もセットにしたいということで検討しています．上流のCASEツールもそうした俎上にのるようなフェーズになってきた．先日ジェームス・マーチン・ジャパンという日本法人と住商情報が共同で，上流工程のCASEツールのセミナーをやりましたが，1000人ほどの会場が満杯でした．それほど皆さんがそこに注目しているんですね．ただ，問題もあります．先ほど**馬場**さんが言いましたが，GUIで見えるがために，それでできたような気になってしまうというところがある．数年前から何回かあったんですが，Access*8で表の項目を入れて，ぺたぺたと貼りつけてGUIができて，それでデータベース設計ができたと思ってしまうんです．実際に動かしてみると何かおかしい．実は裏側ではめちゃくちゃな表ができているわけです．とんでもないSQLがどんどんできていく．本当はデータベースをきちんと設計しなければいけないのですが，データベース設計が欠落してしまっている．

例えばExcel*9でやるのとAccess，SQL Windows，Power Builderでやるのが同じなんだと勘違いされている．裏側にデータベースがいて，そのデータベースを使っていくというのと，自分の表の中でExcelで作っていくというのは全然違いますが，その違いを見せないほどGUIがよくなっているわけです．それがためにそういう間違いを起こしてしまうという例がある．

だいたい効率が悪いとか動かないという話で来るんです．データベースをどう設計したんですかと聞くと，データベース設計なんかしていません．こんなところから始まって，泥沼に巻き込まれてしまう．

馬場　そういったところが正しく伝わっていない．メインフレームもオープンも，システムを開発する基本的な考え方は変わらないと思います．しかし，GUIなどの便利なことが，きちんとした確認手順をとらなくても大丈夫なのだというふうに勘違いされているような気がします．その結果，できたものが期待値ではなかった．こんなに簡単だと言ったじゃないかということになるわけです．そこで初めてシステムというのは慎重に考えなければいけないということがわかる．

最初だからゆえの失敗例になるのかもしれませんが，そうなってしまうと，極端な話，作り直しになってしまう．最初に考えた半年が，また半年かけてもう一回作り直す．これはかけたコストから何から全部無駄になるわけです．その意味では，設計に対する基本的な考え方は従来と変えてはまずい，変わらないものです．

遠藤　先ほどの上流工程で，当社が考えなければいけないのは，きちっとDOA（データ中心アプローチ）ができるSEをたくさん育てないとクライアント/サーバシステムの開発はうまくいかないということです．これからOOだと言われていますが，OOにいくためにもDOAの技術者をきちっと育てていかないといけないと思います．

もともとメインフレームでは，プロシージャ中心，機能中心で，機能展開してきたわけです．会社の組織でも機能というのはその時々で変わります．機能中心で情報処理システムを作ってしまうと，会社の仕組みがちょっと変わったり，やりたいことが変わったりすると対応できなくなってしまう．それに対してデータに注目したのがDOAです．データというのは，その会社が存続するかぎり，追加とか削除はあるかもしれないけれども，その会社がデータを取り扱っていくかということに関しては普遍的です．したがって，データを中心に置いて，その取り扱い方を分析していって，システムを作る時にデータ中心のシステムを作るわけです．

それをもっと進めていったのがオブジェクトで，データとそのデータに対するかかわり方を一つのオブジェクトとしていくというアプローチです．ある人はオブジェクトはもう研究者の手から生産者の手に渡ったと言い，ある人はまだとんでもない，これでうまくいくかと言っている．そのへんは議論の分かれるところです．ただ，オブジェクト指向は，これからDOA，OOD，Javaに代表されるオブジェクト指向のプログラミング，そこが今後の先行する分野だと思います．しかし足下を固めるという意味では，DOAができるSEをたくさん揃えないと，クライアント/サーバシステムの開発がうまくいかないという感じがしています．

今後のメインフレーム

司会　これまでメインフレームとオープンシステムの比較論をしてきていただいたのですが，この辺で，今後のメインフレーム，大型クライアント/サーバというのはどのように進化していくのか，あるいはどういう方向にしようと考えているのかについてお話していただきたいと思います．

今江　メインフレームをどういうふうにしていこうかというのは，まず当面という意味では，基本的にはオープン化と低価格化という二つがあるかと思います．とくにITAS-CA 3800では，低価格化という意味ではCMOSを使うことによって価格を大幅に下げ，ランニングコストも大幅に下げた．

オープン化という意味では，よく言われるように，一つはインターオペラビリティ（相互に連携して動くという世界），もう一つはポータビリティ（プログラムの移植性）の二つがキーになると思います．いずれにしろクライアントのところはPC化されていく．とくにWindowsの世界になってくるのはここ何年かは変わらない．一方でUNIXもパワーアップしてくる，あるいはインテルに乗っていく．そういうものと共存していかないかぎりは，メインフレームはいくら安くなっても存在価値はなくなるだろう．

そういう意味でオープン化は非常に大切で，その中ではインターオペラビリティが重要になる．かつてはUNIX相手のSQLアクセスをサポートすれば良いという話だったが，いまはWindowsとの連携，例えばODBCのインタフェースを持たせるというところがポイントになる．

その次の段階のシステムというのは，一つの筐体の中にシリーズ2200あるいはAシリーズのプロセッサ部分とインテルのプロセッサ部分があって，インテルの方にWindows NT

あるいはUNIXをのせるものです.

先程オープンのプロダクトは整合性が問題になるという話がありましたが，シリーズ2200やAシリーズをオープン・システムとつなげるといった時に，いろいろな組合わせの可能性がある．例えば，データベースもORACLE，INFORMIX，SYBASE等いろいろありますが，ORACLEのこれこれのバージョンとメインフレームのこのソフトウェアとの組合せだったら，きちんとテストしてあって動きますと言えることが意味あるとするならば，このような形態のシステムはユーザにメリットがあるということになります.

Aシリーズの次のシステムでは，Aシリーズの操作を知らなくてもWindowsの世界でメインフレームが操作できる機能が提供されます.メインフレームが今まで持っているデータの保全性，システムの安定性，あるいは既存の資産を生かしながら，ユーザ・インタフェースは，メインフレームの知識がなくても使えるようなそんな環境になっていくと思います.

いずれにしろ今できている製品の世界というのは，従来のメインフレームのよさ，オープンのよさを組合わせていきましょうというものを目指しているということです.

神谷　いま**今江**さんが言われたのとそう変わらないんですが，気持ちはもうちょっとラディカルな感じを私は持っています.

今までサーバと言うとだいたいメインフレームは除外されていたんですが，片やPCサーバが数百MIPSのものとかECCメモリとか，耐障害性機能をどんどんつけてきている．従来メインフレームは耐障害性に強いですよと言ってきたのが，その技術がPCサーバにもどんどん入ってきているということが一つあります.逆にいうとメインフレームはそういうところと競合していく．ある問題を解くのにどのサーバがいいですかという中で，対等に競合していこうということが狙いとしてあります.

その時にどんな方向でやろうとしているかと言うと，私の考えではHMP（ヘテロジニアス・マルチプロセッサ）という技術で，メインフレームはサーバとして姿を現していくと思うんです.具体的に言うと，セッション層のレベルではクライアントとNTサーバで処理してしまう．そこから上がってくるメッセージ，ないしはトランザクションというレベルではメインフレームで処理する.

要するに，セッション層のところまでのやり方は，従来のクライアントと従来のNTサーバのところで，GUIができました，新しい技術ができましたというところは全部吸収してしまうわけです.そこから上がってくるメッセージないしトランザクションというものは，頑強なメインフレームの中で処理して戻していきますよ．そういうことをいまNTとAシリーズというヘテロジニアスなプロセッサで協調しながら処理していく.

そういうやり方をしますと，Aシリーズから見ると，そことインテグレーションされているNTサーバは手足です.NTサーバからAシリーズを見ると，ネットワーク資源に見えてしまう.そういう関係で，セッション層のレベルで従来のメインフレームとNTサーバをくっつけてしまう．ヘテロジニアスなマルチプロセッサをNTにつないでやったというものです.

その先，そういうものをたくさん結合してしまうことです．これをクラスタリングと呼んでいます.そうすると，あらゆるアプリケーションがどこで動いていても，どこでもバックアップしてくれる．そんな世界が今度できてきます.

その意味では，ヘテロジニアスなマルチプロセッサシステムをNTとメインフレームで考えたり，UNIXとメインフレームで考えたりする.とりあえず今マイクロソフトの市場の中で，マーケット上，NTとのところをにらんで考えている．ヘテロジニアス・マルチプロセッサシステムという技術は，いずれ違うものができてきたとしても，たぶん適用していくだろう.最終的にはそれらをクラスタリングで結んで，アプリケーションがどこのレベルでも動ける，どんな障害に対しても完全なノンストップの世界ができていく.

もう一つ，今のプロプライエタリーなOSをPCサーバの上でエミュレーションするということをやっています．かつそこがマルチプロセッサになりますと，片やNTで動いている，片やAシリーズで動いている．筐体そのものは一つです．NTとして使うとシンメトリックなマルチプロセッサですが，そこにAシリーズを乗せることによって，片方のプロセッサはNTとして動き，片方のプロセッサはAシリーズとして動く．PCそのものがNTになったり，Aシリーズになったりする世界です．これはコストダウンして，今のメインフレームをサーバとして小さなところにまで持っていこうという考え方だと思います．

今江　IBM社が1995年，IBM・PCサーバにS/390のボードを搭載した機種を出しましたね．IBM社の一番の狙いは開発環境の提供だったんでしょうか．

田辺　IBM社のMVSでは1台のメインフレーム上に異なるOSの共存が可能です．

ある意味でのHMEを実現しており，今では別の実現方法が可能になってきているわけです．汎用と，オープンなものと共存という形になってきている．

今江　今までのメインフレームの資産はそう簡単にはなくならなくて，まだまだ使われ続けるのではないでしょうか．ガートナが，2010年頃までは，メインフレームは残っているだろうと予測しています．日本は，米国よりもメインフレームの比重が重い世界ですから，2010年になってもまだ残っているのではないでしょうか．

田辺　私も個人的には新しいものが目先が変わって面白いと思いますが，現状で間に合うものを変える必要もない．すごく改善される要素が加わるとすればやる必要があると思いますが，必ずしもそればかりでもない．それだったら従来のものを十分使いこなすというのは，一つの大切な要素だと思います．ものは継続しながら進んでいるし，新しいものが来たら新しいものをどんどん取り込むというのもおかしい．かといって，古いものに従属するという話ではないと思います．

遠藤　ビジネスの世界だから，お金の問題というのは非常に大きいと思います．お金がどういうふうに流れるか，ものの値段がどのくらいになるかというのは大きな要因で，お金さえ安ければ割り切りが出てきているというのがオープンです．マイクロソフト社のビル・ゲイツは，PCが限りなく安くなったらどうなるんだ，1人1台になって当たり前だね，一家に1台になるよね，という発想をしていたと思う．これからどうなるかというと，ネットワークが限りなく安くなったらどうなるのか，ネットワークが限りなく高速になって，ただに近くなったらどうなるのか．

例えば今のメインフレームのチャネルスピードがネットワークでも出てしまう．しかも値段が限りなく安い．どこまでいってもコストは同じ．NTTのOCNはそれを目指していて，来年からサービスを開始する．そういう方向になった時にどうなるのか．サン・マイクロシステムズ社が言っているネットワーク・コンピューティング，ネットワーク・イズ・コンピュータというのが本当に実現してくるのではないか．

そうなった時に，いろいろなサーバがネットワークの中に必要になる．クライアントもいろいろある．イントラネットのブラウザもある．しかし，ブラウザ側は，先程言ったファット・クライアントの問題があって，そんな大変なものはどこかに任せ，使う人のところは，電話と同じで，持ってきてつないだら使える，電気製品と同じようにコンセントにつなげば使える，という世界が理想型ですね．

そのつけはどこに来るかというと，ネットワークとその裏側にあるサーバ群に来るわけです．そう考えた時に，サーバ群の中に重要な位置づけとして，私はメインフレームが来ると思うんです．その時のメインフレームのイメージというのは，基幹システムもやっているし，部門の重要なサーバにもなっているかもしれない．

例えばJavaの発想は私はすばらしいと思います．Netscapeというブラウザがいま各社OS対応，Windows対応，Mac対応，OS/2対応等になっていますが，Javaはそれを透過

にしてしまう．OSはなんでもいい．Hotjavaでブラウザを作れば，どこでも動いてしまう．Hotjavaでなくても，Javaが実行できるところであればなんでもできてしまう．これは一つの理想型ですね．

かつてメインフレームからユーザは，何かに縛られたくないというのでオープンに行ったと思うんです．実際に行ってみたらいろいろ問題はあるけど，いまは何かやろうと思うとクライアントOS，Windowsに縛られていますね．それに対する一つの答えがJavaです．Javaによって，そういう縛りが解ける．Javaが面白いと思うのは，Javaが動くのはサーバであってもかまわないわけです．

そういう意味でみると，メインフレームの中でJavaの言語が高速に実行できて，しかも信頼性が確保できるとすれば，すばらしいプラットフォームです．資産が保証されて，運用管理もきちっとしていて，そのうえでJavaが動くという環境がもしできるのであれば，すばらしいサーバです．

私はネットワーク・コンピューティングの世界が来ると思うのですが，その時にメインフレームの役割というのは非常に大きなウエートを持っていくと思います．ネットワークの非常に大事なところのサーバとなる．ネットワークの中で何が止まったら一番困るかというと，DNSサーバが止まったら動かないわけです．そのDNSサーバをメインフレームがやってくれるとなったらありがたい．そういったところにメインフレームがあてはめられるようにならなければいけない．メインフレームの中でJavaががんがん動くようにならなければいけない．今のJavaはPCでは重くてとても動かない．それが高速に動くサーバが提供されたら，Javaの言語はそのサーバでがんがん動かせばいい．

そういうネットワーク・セントリック，ネットワーク中心の世界になってくると思いますが，その時にどういうクライアントがあり，どういうサーバがあり，その中でメインフレームの特性を生かしたサーバというのは何なのか．それがこれからのメインフレームのあり方ではないかと私は思います．

神谷　まだそこまでは実装のスケジュールにのっていないですね．たしかに私も同意見です．最終的には，全世界に何百万台のサーバがあったとしても，使う側から見るとただ一つのもので，論理的なもの，仮想化されたものという世界だと思います．それを実現しようとして，いま出ているのがJavaだったりしますが，その基礎技術になっているのは分散オブジェクトだと思います．そのへんがここ1年というレンジで見るとまだ実装の時期になっていない．いずれCORBA準拠とか，マイクロソフトだとOLE準拠，このへんが実装されてくるのが，ここ1年，2年のことだと思います．その後ぐらいにならないと，実装レベルでは，いま**遠藤**さんが言われたような世界にはならない．現時点は，そのスタート台に立ちましたよというところかと思います．

今江　その意味ではちょうど中間的というのか，3層アーキテクチャということが去年(1995年)から話題になっていますね．その前は，クライアント側にアプリケーションを持たせたクライアント/サーバシステムを作ればいいと言われていましたが，それではクライアントがだんだんと重装備になってしまうし，管理も大変だということで，アプリケーションのほとんどはサーバ側に持たせ，クライアントを軽くするというのが3層アーキテクチャの実装方法の一つです．それをもっと進めたのがJavaを使ったシステムだと思いますが，そのもう一歩手前の段階では，3層アーキテクチャを元に，クライアントのPCは軽くしておいて，しっかり安定したメインフレーム上でデータとアプリケーションを持たせたクライアント/サーバ・システムでもいいのではないでしょうか．

まさに第1世代のクライアント/サーバ・システムの見直しが行われつつある現在，メインフレームもPCとの組合わせで，エンタープライズ・サーバとして有効に使えるのではないかと思います．

遠藤　その意味で，新イメージのメインフレーム，ネットワークの中でも新たなサーバ，新たなメインフレームサーバというイメージを強く打ち出した方がいいのかと思います．すごく心配している人は，そういうものを打ち出すということは，メインフレームをやめるのかと思う．だけど，それはメインフレームを止めるのとは違うと思うんです．

神谷　そういう使い方もできますよ．あとはTCP/IPを使って，従来のようなトランザクション処理でやってはどうですかということです．それを同時に持っているわけですから．

馬場　今まではそういったこともできませんでしたからね．端末は専用でなければいけないとか，いろいろなところが全部はずれてきましたからね．しかも，昔から持っている強大な財産がそのまま使える．

遠藤　いずれにしろインターネットはまだまだ吹き荒れると思います．

これでどうなっていくかというと，いま言っているネットワーク・コンピューティング，ネットワーク・セントリックの中心がインターネット技術なんです．その意味で，イントラネットと言われているけれども，企業内のイントラネットサーバ，われわれはIntra-Webという名前をつけたんです．クライアント/サーバシステムに対してIntra-Webシステム，提供する技術がちょっと違う．本当はクライアント/サーバなんですが，あまりにもクライアント/サーバが固定概念になりすぎたから，それをちょっと変える意味でもIntra-Webサーバシステムという名前をつけて，使い分けをしています．

そこにメインフレームがあってもいいんです．近いところの打ち出しとしては，Intra-Webサーバとしてのメインフレームというイメージでもいいわけです．今やサーバは，信頼性があって，処理能力があって，運用管理がきちっとしているものがいいのであって，何でもかんでもUNIXとかPCとか言っているわけではないのです．その上にソフトウェアが乗ってきて，運用管理がきちっとしている．先程言ったようにJavaがそこで動いてくれたり，Webサーバのアプリケーションがそこで動くというのがあると，それで十分なんです．

今江　シリーズ2200でもWebサーバ機能は持つんです．それはだんだんできます．

遠藤　それができますということが大きいのです．一つの事実からすると，それもあるかもしれないけれども，それができますということは使う側から見るとすごく大きいんです．

神谷　今のメインフレームにWebサーバをつけて，そのあとトランザクション処理をするためにも，インターネットそのものをインフラにしようという話があります．インターネットをインフラにしてトランザクション処理をしますというシステムです．ですから，Webと今メインフレームで動いているTPモニターとの間にゲートウェイをかませてというやり方だと思います．

遠藤　Webと既存システムとの連携ということでは，CGI（コモン・ゲートウェイ・インタフェース）という共通のものがありますが，これだとかなり重いということで，各社独自のWebサーバのAPIを設けています．そこは各社それぞれ競合状態で，自分のところはこんなに速いというのがそれぞれ出ている．

その裏側では基本的に何を置いてもいいわけです．そこにTPモニターをかませてもいいし，データベース屋さんはそこにデータベースを乗せてきているし，LotusはNotesを乗せてきている．TPモニター屋さんはTPモニターを乗せてくるでしょう．また，ORACLEも，すぐORACLEを乗せてくるでしょう．だから，インターネットが台風の目なんです．

今江　今までの話は，ネットワーク・コンピューティングの中のサーバとしてメインフレームを使っていけるはずだということですね．ただ，一歩前段階で，クライアント/サーバは今までオープンという感じでしたが，企業内でメインフレームをサーバにしたクライアント/サーバシステムも当然使えて，それがさらにインターネットという広いネットワーク環境，IBM社が言うネットワーク・セントリックな環境でも使われていくということだと思います．

遠藤　ネットワーク・セントリックという言葉はガートナグループも使っていて，一般的な言葉になってきている．

神谷　IBM 社のネットワーク・セントリックは，Lotus Notes が入っていて，Lotus Notes＋インターネットということですね．

PC が急激に広まった，クライアントが急激に広まったというのは，歓迎すべき事態ですね．サーバとしてきちんとしたものを揃えなければいけない．逆にいえば大きなビジネスチャンスが出てきたということでしょう．

遠藤　そうです．

今は Web サーバがあって，それに CGI は普通に入っているみたいですが，それをさらにエンハンスして，独自の API を作って，さらに当社の XIS とつなげたり，LINC とつなげたりというところをやらなければいけないでしょう．そこは今後考えなければいけない．例えば MAPPER を使っているユーザの方は，MAPPER データベースがあるのでしたら，それはそのままお使いください．MAPPER が Web の裏側にいて，Netscape から MAPPER データベースが見られるというのはすばらしいことですね．

今江　当社は，既存のメインフレームの資産を生かしながら，企業レベルのクライアント/サーバ・システムを段階的に構築していく考え方を ClearPath™ として提唱しています．さらに，それを実現するためのプロダクトとサービスを ClearPath ソリューション・セットとして提供していますが，その一つとして Web サーバとの連携を可能にするような新しい形態のものも準備していきたいと思っています．

神谷　ClearPath はつなぎの名前で嫌だという話がありますが，そういうイメージもあるのかも知れませんね．だけど私は ClearPath というのはたえずわれわれのシステムの目指す明確な道筋を示すものであり，それを達成するプラットフォームがあり，プロダクトがありという解釈をすればいいと思います．毎年中身の顔は成長していくと思っています．確かに今の時代だから NT を意識しました．でも，5 年後にはまた違ったそういうものがあったら，HMP 技術を適用して統合すればいいわけです．

司会　本日は多くのテーマについて語っていただくことになり，個々のテーマについて掘り下げが十分にはできませんでしたが，どのような課題が存在するのかは読者の方にご理解いただけたかと思います．

長時間ありがとうございました．

＊1　WindownNT は，Microsoft 社の商標である．
＊2　UNIX は，X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
＊3　INFORMIX は，INFORMIX Software 社の登録商標である．
　　SYBASE は，Sybase 社の登録商標である．
　　ORACLE は，Oracle 社の登録商標である．
＊4　DOS/V は，IBM 社の商標である．
＊5　SPARC は，SunMicrosystem 社の登録商標である．
＊6　NetWare は，Nobell 社の登録商標である．
＊7　PowerBuilder は，Powersoft 社の商標である．
＊8　Access は，Microsoft 社の登録商標である．
＊9　EXCEL は，Microsoft 社の登録商標である．
＊10　Mac は，Apple Computer 社の登録商標である．
　　OS/2 は IBM 社の商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

分散システム管理ソフトウェア「DSmgr+」

1. はじめに

「DSmgr+（ディーエスマネージャプラス）とは何？」と思っている読者の方に，一言で説明させて頂くとすれば，それは「UNIX*1 と PC で構成される分散システムを統合的に運用管理するための製品である」ということになるかと思う．ポイントは，PC と UNIX が混在した分散システムを，どう運用し，どう管理するかであり，DSmgr+ では，オブジェクト指向管理基盤をベースに，マネージャ/エージェント方式によって分散システムを効率的に運用管理している．

2. DSmgr+ 誕生の背景

近年，世の中の風潮として「オープン化」「ダウンサイジング化」が叫ばれ，マルチベンダによる分散システムの構築が急速に広がってきている．それに伴い，システムの運用管理はそれまでと比較にならないほど複雑化してきた．そこで「マルチベンダ環境下の情報ネットワークを，統合的かつ円滑に運用管理するための支援システム」が求められ，日本ユニシスは，1994 年 2 月に「ASMF (Advanced Systems Management Framework)」という管理体系を発表し，このニーズに応えた．

「ASMF」は，その管理目的により四つに分類された管理分野と一つの統合管理ビューから構成されている．それぞれ，「プラットフォーム管理」「アプリケーション管理」「ネットワーク管理」そして「分散システム管理」である．

これら管理分野の一つである「分散システム管理」の中核をなす製品として「ASMF」と同時に DSmgr を商品化した．

このときは，UNIX プラットフォームだけで，管理アプリケーションも基本的な 3 機能つまり，構成情報を管理する DSmgr/コンフィグ，障害を管理する DSmgr/フォールト，それにソフトウェアを配布する DSmgr/リリースのみであった．

その後，世の中では高機能化・高性能化・低価格化が進んだ PC の普及が目ざましく，能力的にも UNIX ワークステーションに匹敵するレベルにまで近づいてきた．そして，その低価格さが拍車をかけ，PC の大量導入に繋がり，結果として企業システムの規模が一気に膨らむことになる．

従来の UNIX だけのシステムでは数十台～数百台．どんなに多くとも数千台規模であったものが，PC がその管理対象に入るとなった途端に数万台の規模にまで拡張していく．最近は，全社員を対象に PC 1 人 1 台設置という企業も増えてきた．従って，それらを想定した大規模システムへの対応は，これからの企業の運用管理を考えると必須と言えるであろう．また，管理機能の追加やプラットフォームの拡張も併せて求められてきた．

これらの業界動向やユーザ・ニーズに対応し，従来の DSmgr を包含した形で今回新たに「DSmgr+」として装いも新たに商品化した．

3. DSmgr+ の特徴

したがって DSmgr+ の特徴としては，U 6000 や US ファミリおよび Sun (Solaris)/HP 等の各 UNIX プラットフォームから WindowsNT*2 および Windows 95 を搭載した PC までのマルチベンダ/マルチプラットフォームへの対応や，1 台構成から数万台までの大規模への対応といった点が挙げられよう．

また，画面（定義情報を作成するための入力画面や，実行状況の把握画面）に表示される内容と操作が DSmgr+ の各製品全体を通して全く同一形式だという点，つまりユーザにとって一番大事な見た目と操作が統一されている点，これは DSmgr+ の隠れた最大の特徴というか魅力である．

一方，今後の技術革新にもスムーズに対応できるよう，オブジェクト指向技術の採用や国際標準/業界標準への準拠等を行っている．

4. DSmgr+ の管理機構とシステム構成

DSmgr+ での管理の仕組みは，マネージャ（管理する側のアプリケーション群）とエージェント（管理の対象となる機器）間での操作指示と通知の

相互伝達という方式により運用管理を実現している．つまり，マネージャから指示を出し，エージェントで実行し，その経過や結果および進捗をマネージャに通知するという仕組みである．

このような，一つのマネージャと一つ以上のエージェントからなる領域を，分散システムを効率的に運用管理するための管理領域単位と呼ぶ．

一つのマネージャで管理できるエージェントの数は，企業の規模や構成，地域性，運用形態，稼働させるアプリケーション等により様々で，すべてを一つの管理領域に設定することも可能であるが，複数の管理領域に分割した方がいい場合もある．そのような場合，複数になった管理領域をまとめて操作・管理・運用・監視を行う「マネージャ・オブ・マネージャ」を設けることができる．これにより，管理の基本単位であるマネージャ/エージェントという2階層の管理構造から3階層の管理構造をとることが可能となり，小規模から大規模までのシステム構築が容易にかつ柔軟に行えることになる．したがって，支店や営業所単位に一つの管理領域を設定し，本社や管理センター等で全体を管理することが可能となる（図1）．

5. DSmgr+の製品構成および主な機能

DSmgr+各製品の全体概要を図2に示す．本稿では主な機能についてそのポイントを紹介する．

1) DSmgr/コンフィグ

① ハードウェアの構成・配置・属性情報を統合的に管理する製品であり，ホスト名やアドレス等の構成情報の登録・追加・削除・更新・検索を，DSmgr/コンフィグが提供しているグラフィカルな画面を使って簡単に行える．DSmgr+を使用する上で必須の製品である．

② ツリー構造によるグルーピング管理が可能で，管理領域単位やネットワーク単位，セグメント単位で論理的にグループ化することにより，(アイコン化して)全体の管理をより容易に行える．

③ ネットワーク構成図と配置図および接続図を表示させる事により，システム構成(実際の接続形態)をビジュアルに把握できる．

2) DSmgr/フォールト

① エージェントから通知される多くのメッセージを集中的に管理し，リアルタイムに監視画面に表示する．

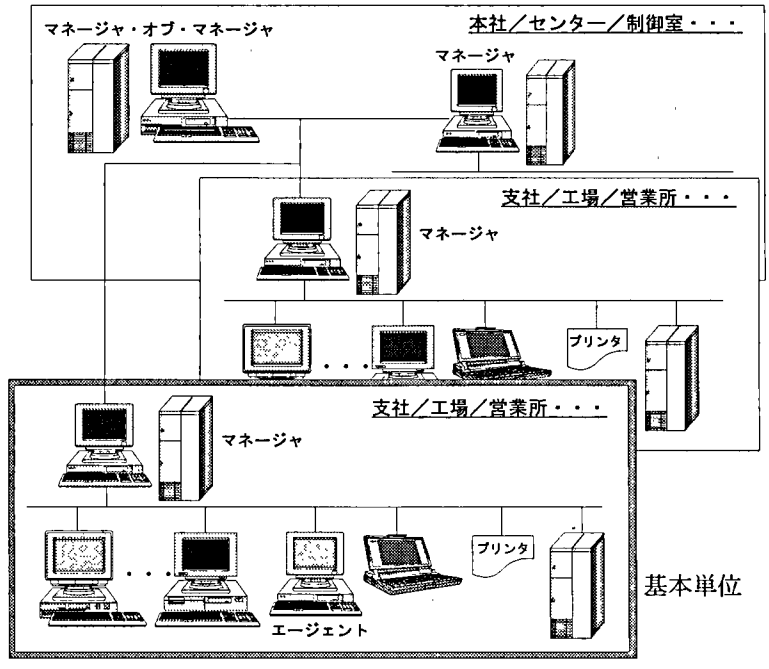


図1 DSmgr+のシステム構成

② DSmgr/コンフィグと連動し，ネットワーク構成図中の障害発生箇所のアイコンの色を赤く変えることにより，障害状況を管理者に通知する．

③ 「しきい値」により，ユーザ独自の管理境界値を設定することができ，かつその監視と報告も可能である．

④ DSmgr/フォールトが提供するAPIを使って，ユーザ・プログラムから任意のメッセージをDSmgr/フォールトに通知することが可能である．

⑤ 障害発生時，自動的に実行するプログラム(シェル)を指定できるスケジュールド・タスクの登録や，障害の症状/原因/対策等の説明文を付加情報として加えたトラブル・チケット等により障害回復を支援することができる．

3) DSmgr/リリース

① ソフトウェアやデータ・ファイルの配布および入れ替えまでを，進捗と履歴を含めて統合的に制御/管理する．

② 配布処理のための申請手続きは，グラフィカルな画面で簡単に行える．

③ 配布には，指定日時に自動的に配布する予定日配布や，ユーザが手動で配布できる機能があり，圧縮して転送する．

④ ブート時入替/シャットダウン時入替/ユーザ起動入替等，運用に合わせた入替処理を選択できる．

⑤ 配布/ローディング/入れ替えまでを一気に実行できる「緊急配布」機能や入替え後に起こった様々な障害や配布ミス等により，入替え前の状態に戻す必要が生じた場合に元の環境（一世代前）に戻すことができる「配布解除」機能がある．

4) DSmgr/バックアップ

① スケジューリングによる自動バックアップやジュークボックスのスロットを指定したテープのサイクル運用/クリーニング・テープの運用支援等により，大規模システムにおける無人運転を可能としている．

② レベル指定による効率的なバックアップやオンライン・インデックスの利用による迅速なリストアを実現．また，クライアントをグループ化することにより，サーバにかかる負荷を軽減できる．

③ 各サーバからの集中管理や操作および履歴管理と的確な状況監視が可能である．

5) DSmgr/ジョブ

① メインフレームの世界で実現していた自動ジョブ・スケジューリング等の運用管理機能をUNIX上で実現したもので，バッチ・ジョブの登録から実行/監視に至るまでを集中的に統合管理する．

② ジョブの実行順序をジョブ・ネットワークとしてフローチャート形式で定義でき，そのジョブ・ネットワーク図を利用して，ジョブの進捗状況をビジュアルに監視できる．

③ 部門ごとにも設定可能なジョブ実行カレンダやイベントによるジョブ・スケジュールの登録と自動起動ができる．

④ 強制実行/強制終了/再実行等，ジョブ・リカバリのための多彩な実行制御を提供．

6) DSmgr/パフォーマンス（平成8年9月提供予定）

① CPUやメモリの使用率，ディスクの空き容量等の性能データを測定，集計，表示および多彩な形式でのレポート出力が可能である．

② 独自のレポート形式を持つユーザへの配慮として，市販の表計算ソフトで集計/加工ができるようなファイル形式での出力も可能としている．

③ 測定項目ごとに「しきい値」の設定が可能であり，DSmgr/フォールトと連携することにより監視画面への警告通知ができる．

7) DSmgr/アカウント（平成8年9月提供予定）

① システム資源の使用量に応じた課金処理を統合管理し，簡単な課金条件の設定で豊富な課金レポートの出力ができる．

② 市販の表計算ソフトで集計/加工ができるようなファイル形式での出力も可能．

8) SMS連携機能（平成8年11月提供予定）

① Microsoft社のSystems Management Serverと連携し，SMS配下のPCを統合管理する．

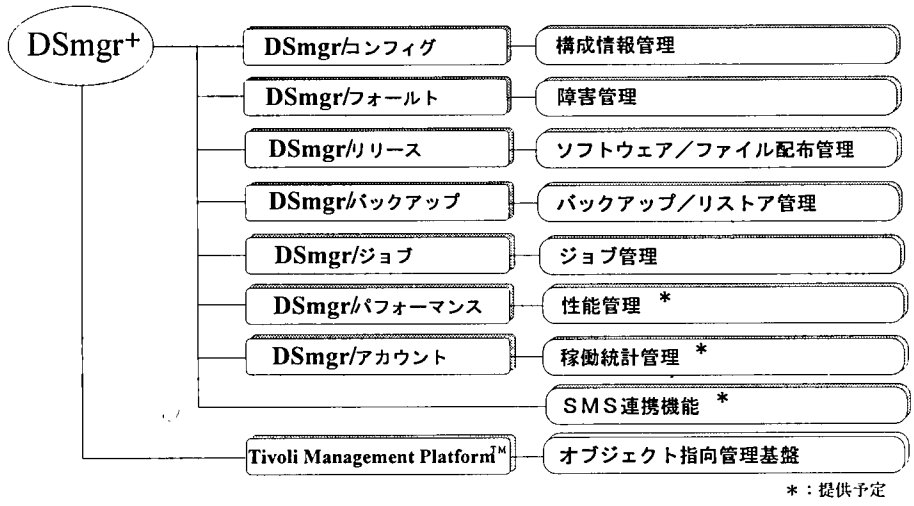


図 2 DSmgr+ の製品構成

6. 今後の展望

PCがこれだけ普及し，NTサーバを中心としたシステム構築が増加する傾向にある今日，NTサーバへの対応は必然的とも言える．当社は，まず手始めにSMS連携機能を提供する．これは，すでにUNIXが設置されている所へ，PC（Windows NT）を入れようとしているユーザで，PCの管理はSMSでやりたいが，UNIXも含めた企業全体の運用管理を求めるユーザに有効である．PCネットワークの運用管理であるSMSと連携することにより，互いの特徴・特異性・利点を十分生かした運用システムの構築を目指していくのがユーザにとっても得策と思われる．次に，管理アプリケーションの適用がある．バッチ・ジョブの管理やバックアップ，性能管理といったUNIXサーバを中心に運用している管理機能をNTサーバに対応する．最後はメインフレームも含めた他の管理システムとの連携である．とくにネットワーク管理システムとの連携が今切望されている．ともかく，「分散システムの運用管理はDSmgr+におまかせ！」という心構えで随時機能の拡張と充実を図っていく．

＊1 UNIXは，X/Openカンパニーリミッドがライセンスしている米国ならびに他の国における登録商標である．

＊2 WindowsNTは，Microsoft社の商標である．

その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

お詫びと訂正

前号（1996年2月発刊 Vol. 15 No. 4）で下記のとおり誤りがありましたので，お詫びして訂正いたします．

p. 40 （誤） A Method for Determining the of Tangent Lengths of Ferguson Curves

（正） A Method for Determining the Tangent Lengths of Ferguson Curves

p. 42 （誤） $B_{1,n}(t)={}_nC_1(1-t)^{n-1}t^1$ (2-3)

ここで ${}_nC_1$ は，！を階乗の記号とするとき

${}_nC_1=n!/(i!(n-i)!)$ (2-4)

（正） $B_{1,n}(t)={}_nC_1(1-t)^{n-1}t^1$ (2-3)

ここで ${}_nC_1$ は，！を階乗の記号とするとき

${}_nC_1=n!/(i!(n-i)!)$ (2-4)

p. 105 （誤） An XTPA System for Muoti-user Sites

（正） An XTPA System for Multi-user Sites

掲載論文梗概

シリーズ2200のオペレーティング・システム(OS 1100)は，UNIVAC 1107のEXEC Iを起源とし，本年(1996)まで34年間の歴史を持つ．伊藤龍彦は**シリーズ2200オペレーティング・システムの発展**の中で，シリーズ2200のオペレーティング・システムの系譜を辿り，基本設計と改良のポリシーとを概観している．

Aシリーズのオペレーティング・システム MCPは，「資源の仮想化の実現」を基本に据えて開発された．一方，プロセッサ/入出力機器の高速化とメモリの巨大化の急速な進展の中，新たな技術の開発と実装を行い，大容量データベース・大規模トランザクション処理を実現してきた．そして今日，オープン環境での異種システムとの接続と「異なるオペレーティング・システムとの統合」技術を実装する段階に入った．鈴木秀明は**Aシリーズオペレーティング・システムの発展**の中で，MCPおよびAシリーズで実現してきた技術の変遷を述べるとともに，今後のオープン・エンタープライズ・サーバを支える技術を展望している．

ITASCA 3800シリーズ・システムは，高性能な0.5ミクロンCMOS標準セル・チップの先進ファミリの基盤技術をユニシス独自のロジックとカスタム分散アレイ技術に結び付けて実現されたものである．**ITASCA3800シリーズのハードウェア技術**は，まずITASCA 3800のハードウェアの特長を，次にITASCA 3800の構成要素についてハードウェア技術の面から詳細に解説している．

討論会「**エンタープライズ・サーバの現状と展望**」は，6人の出席者のオープンとメインフレームの立場での座談会形式による発言を通し，メインフレームとオープンシステムの特質について明らかにし，それぞれの適用の方向性を探ってみようとしたものである．

*1 UNIXはX/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．
*2 Windowsは米国Microsoft社の登録商標である．
その他，本稿に記載の会社名，商品名は一般に各社の商標または登録商標である．

掲載論文梗概

ISSN 0914-9996

技　報
UNISYS TECHNOLOGY REVIEW
Vol. 16　No. 1　(No. 49)

発　行　日　平成8年5月31日
編集発行人　吉　村　賢　一
発　行　所　日本ユニシス株式会社
東京都江東区豊洲1-1-1 〒135
TEL(03)5546-4111 (大代表)
印　刷　所　三美印刷株式会社