# ネットワークＯＳの設計に関する一考察

松山 晃久　白鳥 則郎　野口 正一

（東北大学 電気通信研究所）

## １．まえがき

知的な分散処理システムを構築するためには、次の３つの技術が必須のものとなる。ここで、知的な分散処理システムとは、ユーザに知的な利用環境のもとで、時間的及び空間的制約を意識させずに、散在する多種多様なリソースの使用を可能とするシステムである。

(1) マンマシンインターフェース
(2) 情報ネットワーク
(3) リソースの統合化

本稿では、(2) の技術で必要となるソフトウェア、特にネットワークOSの設計問題について考察する。情報ネットワークを構成する上で必須のソフトウェアは、ネットワークOSと通信ソフトウェアである。前者は、主としてユーザに情報ネットワークを仮想的な１つのシステムとしてみせるための機能を提供する。また、後者は通信プロトコルを具体的に実現するためのソフトウェアである。

ネットワークOSの設計には２つの立場がある。ひとつは、既存のOSの上にネットワークを制御するための機能を付加する方式である。もう一つは、既存のOS機能も包含するものとして全く新しくネットワークOSを構成する方法である。これらの方式には、共に長所と短所が混在している。従来、前者の立場で設計する方式がほとんどであった。この方式の長所は、既存OSを意識して設計するため、実現が比較的容易である。一方、既存OSを意識するあまり、汎用性に乏しい傾向があった。

本稿では、新たに既存のOSの機能を含むネットワークOSを設計する立場からネットワークOSの論理構造を明確にし、同時に、汎用性を指向する設計法について検討する。

また、この設計法に基く具体的なネットワークOSを試作し、実証実験し、その有効性を示す。

## ２．ネットワークＯＳの論理設計

### ２．１ 設計の基本概念

(1) 仮想機械とその分解

NOS の基本的な機能は、通信回線などで結合された多種多様な情報資源を時間的及び空間的な制約なしに、ユーザに提供することである。つまり、ユーザがシステムを１つの仮想機械 VM(Virtual Machine) として扱い、使用できる環境を実現することである。

一般に、システムの規模が大きくなるにつれ、その機能を１つのVMで実現することは、記述性、理解性などの観点から難点がある。そのため、本稿では、"VM の分解" を導入する。つまり、図 2.1に示すようにトータルシステムの機能に注目し、段階的にVMを分解する。これは、設計の詳細化にも対応している。まず第一段階の分解は、ネットワークを構成している各ホストを$VM_i$ としてとらえ、それぞれの$VM_i$ が互いに関連を持ちながらネットワークとしての機能を実現して行くと考えることができる。各ホストである$VM_i$ は更にいくつかの$VM_{ij}$に分解することができ、これが更に$VM_{ijk}$ に、というようにシステム設計者が設計しやすいようにどこまでも分解して行くことができる。

本論文では、図 2.1における第 2段階までの分解を出発点としてNOS 設計を開始することにする。このような考えに基くと、設計とは、各VMの内容とVM間の論理関係を実際に規定することである。

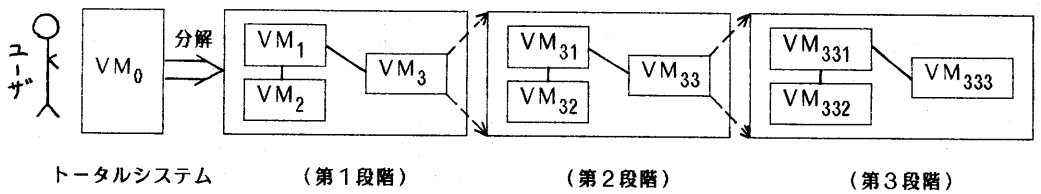


図２．１ システムのモデルと仮想機械

(2) VMの種類

あるVMが他のVMを「使用」するために、VM間で情報交換が行なわれる。ここで、VMは次の２つのタイプに分類される。

i ) requester VM
ii) server VM

server VM の動作はrequester VMからのリクエスト情報の系列によってのみ決まるのに対し、requester VMの動作は内部に持っているプログラムによって決まると考える。ここでは、システムは、これらのrequester VMとserver VM の集合であると考え、モデル化を試みる。

(3) VMの階層化

システムを構成するVMに、階層性を導入する。図 2.2に示すように、ユーザは与えられたコマンド集合{ $UC_i$ } を発する。そのコマンドに対する動作を行なうために必要なrequester VMがまず起動される。これが動作して行く過程で必要なサービスを与えるserver VM があり、それに対して更にサービスを与えるserver VM が存在する。これらのVMのうち、server VM だけを集めたものをNOS としてとらえることができる。それに対してrequester VMはアプリケーションである。

## ２．２　ＮＯＳに要求される機能

ネットワークトランスペアレンシを持ったNOSを構築するためには、以下にあげるような機能が必要となる。

(1) ネットワークを意識する必要のないプロセス間通信機能
(2) プロセスを必要なホスト上へ起動する機能
(3) ネットワークワイドなネーム管理機能
(4) ネットワークワイドなファイルシステム機能
(5) ネットワークを意識する必要のないユーザインタフェースの提供

## ２．３　論理設計

前述したNOS の機能を実現するために必要となる基本機能として、ここでは、次の２つを導入する。

(a) 任意のホスト上のファイルをアクセス(Read, Writeなど) する機能
(b) 任意のホスト上にプロセスを起動する機能

例えば、あるデバイス上のファイルをあるデバイスへ転送するコマンドを考える。これを実現するには、このコマンドに対応したアプリケーションプロセス(AP)を作成する。このAPの動作は、前記の基本機能(a) を用いて、指定のファイルを読み、次に目的のデバイスへそのファイルを書き込めばよい。また、自分のホスト上にコンパイラがない場合、ソースプログラムとデータを目的のホストに転送し、コンパイラを起動させればよい。

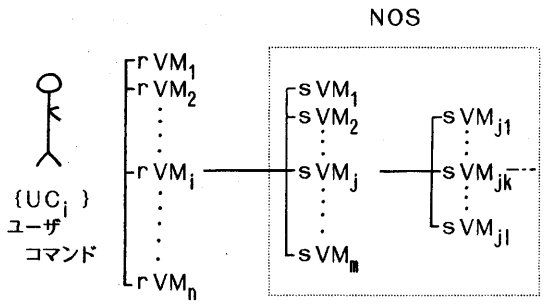


図2．２　VMの階層構造化

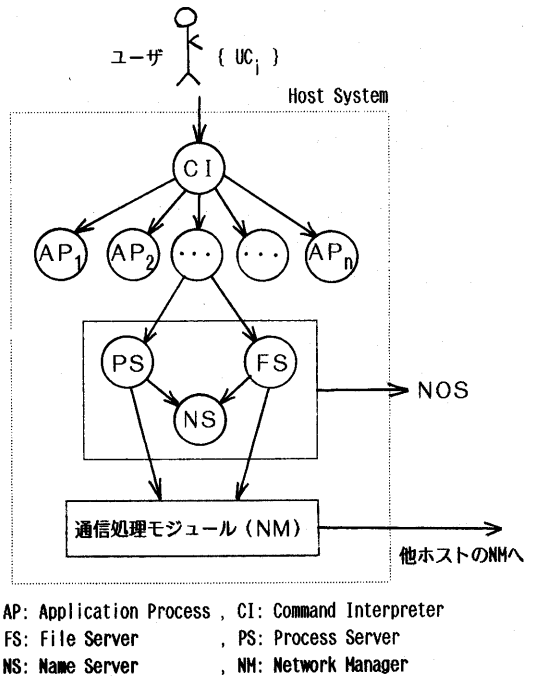


図2．３　ＨｏｓｔとＮＯＳの機能モジュール

基本機能(a) と(b) に対応し、プロセスサーバ(PS)とファイルサーバ(FS)を、またNSとNMなどを以下のように定義する。

(1) PS( プロセスサーバ )

任意のホスト上に指定したプロセスを起動させる機能モジュール。指定したプロセスが他のホストにある場合、そのホストのPSを通して起動させる。

(2) FS( ファイルサーバ )

ファイルに対する Open, Close, Write, Readを実行する機能モジュール。ここで、ファイルはデバイスも含む。

(3) NM( ネットワークマネージャ )

任意のホスト間におけるプロセス間の通信機能を提供する機能モジュールである。OSI 参照モデルと対応させると、セッション層以下のプロトコルを実現する通信処理プログラムである。本稿では、NMの構成については立ち入らない。

(4) NS( ネームサーバ )

ネットワーク中の資源の名前、所属するホスト名、属性などを管理する機能モジュール。

(5) CI( コマンドインタプリタ )

ユーザコマンドを解釈し、対応するAPを起動させる。

(6) AP( アプリケーションプロセス )

PS, FS, NSやNMを用いてユーザコマンドを実現する機能モジュール。

以上の機能モジュールを用いてホストの論理構造を設計したものが図 2.3である。

## 3. ネットワークOSの仕様記述

### 3.1 記述法

2節で述べたNOS の構造にうまく適合し、これを反映した記述形式として、本論文では、その構成要素であるVMを有限オートマトンを用いて表現する。一般にシステムをVMの集合と考えると、その仕様記述は次の2つを規定する問題に帰着する。

①各VMの記述
②各VM間の論理関係の記述

VMに対する入力として、他のVMからの信号( これをイベント信号と呼ぶ) とプログラムの各命令( ステップと呼ぶ) を考える。また、VMの出力として、VM内のプライベートな変数を変化されること。及び他のVMへの信号の送出という動作を対応させる。これらの詳細については後述する。

このような性能をもつVMを表現するオートマトンとして「ユーザマシン」と「システムマシン」の2種類を導入する。

### 3.2 ユーザマシンとシステムマシン

(1) ユーザマシン(UM)

これは入力として、イベント信号とプログラムの各命令 ( ステップ ) の2種類があるオートマトンである。ここで、プログラムは以下で定義する入力ステップを用いて書かれている。プログラムはオートマトンを動作させるためのスケジューラの役割をする。

ユーザマシンを次式で定義する。

$$UM = < K, \Sigma_e \cup \Sigma_s, \Delta, \delta, \omega, q_0 >$$

$K$ : 状態集合
$\Sigma_e$ : 入力イベント集合
$\Sigma_s$ : 入力ステップ集合
$\Delta$ : 出力集合
$\delta$ : 状態遷移関数
$\omega$ : 出力関数
$q_0$ : 初期状態

(2) システムマシン ( SM)

入力としてイベント信号だけのオートマトンであり、スケジューラを持っていない。従ってシステムマシンは他のオートマトン ( ユーザマシン、システムマシン ) に「使用される」という性質を持つ。システムマシンを次式で定義する。

$$SM = < k, \Sigma_e, \Delta, \delta, \omega, q_0 >$$

( 記号の意味はユーザマシンの場合と同様である。)

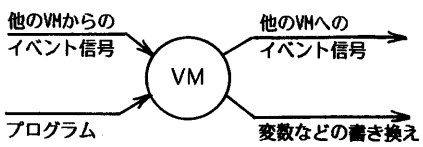


図3.1 VMの入出力の種別

以上の定義に基づいて、NOS の各VMを記述する。APの記述は、ユーザマシンとそれに対するプログラムを記述することにより与える。

AP＝＜ユーザマシン、プログラム＞

システムプロセスの記述には、システムマシンを記述することにより与える。

SP＝＜システムマシン＞

以上述べたことを、２．と合わせて整理すると図 3.2のようになる。同図においてユーザがコマンドを出すと、コマンドインタプリタCIがそれを受けとり、コマンドに対応するAPを起動する。すると、ユーザマシン内の命令カウンタが、APプログラムの最初の命令にセットされ、そのステップが命令カウンタの増加とともに順次ユーザマシンに入力され、ユーザマシンは、それに応じた動作をし、目的の仕事をしてゆく。イベント信号を受信したVMも更に同様の動作をして、APに対してサービスを行ってゆく。APプログラムの最後まできて、もうステップがなくなると、APは消滅する。

## ３．３　各ＶＭの記述

ここでは、APの例として、TYPEコマンドと、それに関係するシステムモジュールの記述を行う。TYPEコマンドはネットワーク中の任意のデバイス上のファイルを任意のデバイスへ印字する動作を行うコマンドである。

TYPEコマンド；TYPE（ソースファイル名，
ディスティネーションファイル名）

AP(TYPE)＝＜TYPEユーザマシン，TYPEプログラム＞

ここで、AP(TYPE)はTYPEコマンドに対応したAPを示す。このAP(TYPE)は、 2.3で導入した機能モジュールFS,NS,NMなどを用いて前述の機能を実現する（図 3.3）。

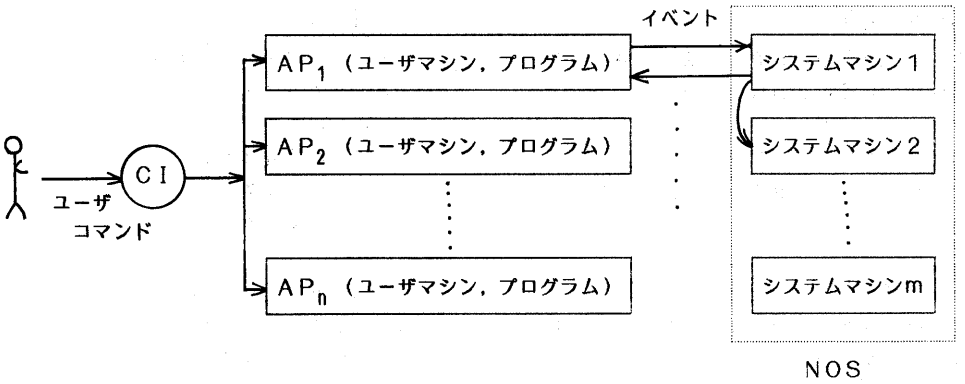


図３．２　VMを用いたＨｏｓｔの論理表現

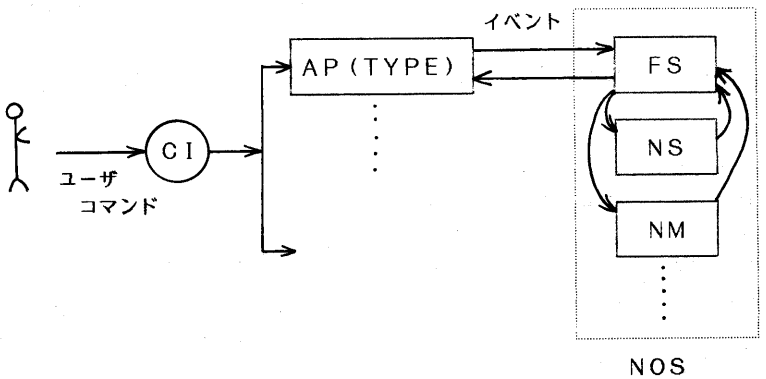


図３．３　ＴＹＰＥコマンドに対応するVMの関係

AP(TYPE)は上記で定義したように、①TYPEユーザマシンと②TYPEプログラムを規定することによって構成される。まず、AP(TYPE)を記述、つまりTYPEユーザマシンとTYPEプログラムを記述し、次に、AP(TYPE)が使用するシステムマシンであるFSシステムマシンとNSシステムマシンを記述する。

TYPEプログラム

read(filename)
write(filename)

TYPEユーザマシン

・オートマトン名　UM=TYPE
・状態集合　$K=\{S_0, S_1, S_2, S_3\}$
・入力集合　$\Sigma=\Sigma_s \cup \Sigma_e$
　ステップ　$\Sigma_s$ ={read(filename),write(filename)}
　イベント　$\Sigma_e$ ={f#,data}
・出力集合
　Δ={event(open(filename),FS),event(read(f#),FS),
　　buf ←data,event(close(f#),FS),
　　event(write(f#,buf),FS)}
・状態遷移関数δ
　$S_0$ ×read(filename)→ $S_1$
　$S_1$ ×f#→ $S_2$
　$S_2$ ×data→ $S_0$
　$S_0$ ×write(filename) → $S_3$
　$S_3$ ×f#→ $S_0$
・出力関数ω
　$S_0$ ×read(filename)→event(open(filename),FS)
　$S_1$ ×f#→event(read(f#),FS)
　$S_2$ ×data→ buf←data;event(close(filename),FS)
　$S_0$ ×write(filename) →event(open(filename),FS)
　$S_3$ ×f#→event(write(f#,buf),FS);event(close(f#),FS)
・初期状態　$q_0$ =$S_0$
・内部変数　buf

FSシステムマシン

・オートマトン名　SM=FS
・状態集合　$K=\{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$
・入力イベント集合
　Σ={open(filename,read(f#),write(f#,data),
　　close(f#),local,remote,data,f#}
・出力集合
　Δ={event(ask(filename),NS),f#←f#+1, event(f#,
　　呼出元オートマトン),event(data, 呼出元オートマトン),
　　f#←f#-1,event((remoteFS,open(filename)),NM),
　　event((remoteFS,close(f#)),NM),
　　event((remoteFS,read(f#)),NM),
　　event((remoteFS,write(f#,data)),NM),localopen(f#),
　　localclose(f#),localread(f#),localwrite(f#,data)}
・状態遷移関数δ
　$S_0$ ×open(filename)→ $S_1$
　$S_1$ ×local → $S_2$
　$S_2$ ×read(f#)→ $S_3$
　$S_2$ ×write(f#,data)→ $S_3$
　$S_3$ ×close(f#) → $S_0$
　$S_1$ ×remote→ $S_4$
　$S_4$ ×f#→ $S_5$
　$S_5$ ×read(f#)→ $S_6$
　$S_6$ ×data→ $S_7$
　$S_5$ ×write(f#,data)→ $S_7$
　$S_7$ ×close(f#) → $S_0$
・出力関数ω
　$S_0$ ×open(filename)→event(ask(filename),NS)
　$S_1$ ×local →f#←f#+1;localopen(f#)
　　;event(f#,呼出元オートマトン)
　$S_2$ ×read(f#)→localread(f#);
　　event(data, 呼出元オートマトン)
　$S_2$ ×write(f#,data)→ localwrite(f#,data)
　$S_3$ ×close(f#) → localclose(f#);f#←f#-1
　$S_1$ ×remote→event((remoteFS,open(filename)),NM)
　$S_4$ ×f#→ event(f#,呼出元オートマトン)
　$S_5$ ×read(f#)→event((remoteFS,read(f#)),NM)
　$S_6$ ×data→event(data, 呼出元オートマトン)
　$S_5$ ×write(f#,data)→event((remoteFS,write(f#,data)),NM)
　$S_7$ ×close(f#) →event((remoteFS,close(f#)),NM)
・初期状態　$q_0$ =$S_0$
・内部変数　f#(初期値0)

NSシステムマシン

・オートマトン名　SM=NS
・状態集合　$K=\{S_0\}$
・入力イベント集合　$\Sigma_e$ ={ask(filename)}
・出力集合
　Δ={search(table),event(local,呼出元オートマトン),
　　event(remote, 呼出元オートマトン)}
・状態遷移関数δ
　$S_0$ ×ask(filename) → $S_0$
・出力関数ω
　$S_0$ ×ask(filename) →search(table);
　　event(local,呼出元オートマトン);
　　event(remote, 呼出元オートマトン)
・初期状態　$q_0$ =$S_0$
・内部変数　ファイル名管理テーブル

## 4．　ネットワークOSの試作

### 4．1　実験システム

前章で述べたNOS の仕様記述に従って、実際にNOS を図 4.1に示す実験システム上にインプリメンテーションを試みる。

実験システムは２台のパソコンをRS232Cチャネル（調歩同期，4800bps)を通じて結合されており、それぞれ既存OSとしてCP/Mが働いている。NOS の各VMはこのCP/Mの機能を一部利用して実現することになる。

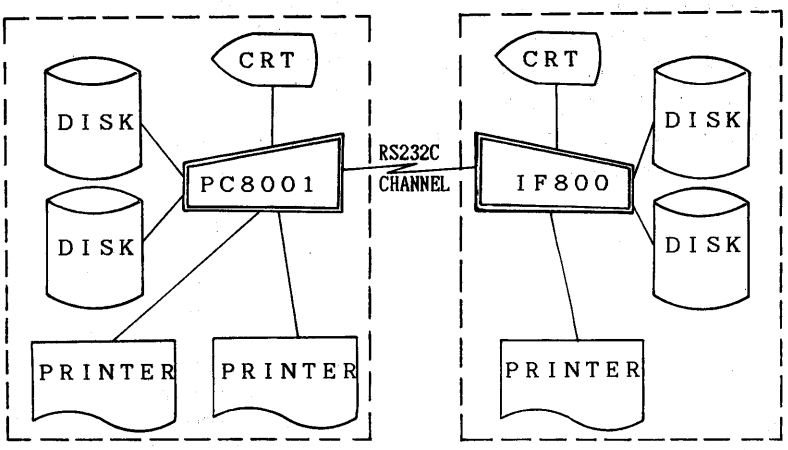


図4．1　実験システムの構成

このシステム上において実行できるユーザコマンドとして次の２つのコマンドを実行するAPを作成する。

(1) TYPE [dev1:]filename dev2:[filename2]
任意のデバイスdev1のファイル名filename1 を任意のデバイスdev2へ出力するコマンドである。

(2) EXEC [dev1:]sourceprogram [dev2:]datafile
Fortran ソースプログラムをコンパイル，リンケージを行い実行する。

上記の２つのコマンドの[　　]内は省略可能である。即ち、ファイル，コンパイラ等の存在場所をユーザは知らなくともよい。

## 4．2　CP／Mを用いたNOSの作成

VMを既存OSを利用して実現する場合、既存のOSにより実現できる部分とNOS 用に新たに作成する部分の境界をはっきりさせなければならない。これは、前章で行った仕様記述において出力関数を記述するために用いた記述用関数のうち、いくつかはCP/Mの機能に対応するように記述してあるので、その他の部分を新たに追加作成すればよい。

図 4.2に１つのホスト上のメモリ構造を示す。この図において、F80 はFortran コンパイラ、L80 はリンケージローダ、UVM はユーザのFortran で書いたソースプログラムがF80 、L80 によって変換さ

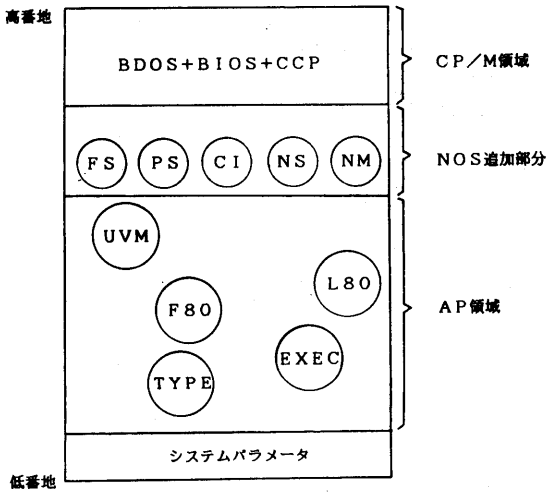


図４．２　メモリ構造

れて新しくできるVMである。図 4.2ではNOS 各モジュールとCP/Mを分離して示してあるが、先に述べた通り、論理的には各モジュールの中にCP/Mの機能が組込まれている。プログラムはアセンブリ言語(MAC-80)を用いて記述し、NOS の追加分(FS,PS,CI,NS,NM)のサイズは、PC8001用とIF800 用ではそれぞれ 550ステップと 300ステップである。また、APであるTYPEとEXECのプログラムサイズはそれぞれ 150ステップと 300ステップとなっている。

## 4．3　動作実験

試作したシステムの動作例を図 4.3に示す。ここでは、最初にTYPEコマンドによりDSKIFA上のファイルSIN.FOR の内容をCRTPC へ印字している。SIN.FOR の存在場所がわからなければDSKIFA: は省略しても同様の結果である。次のEXECコマンドでは、ユーザのいるホスト上にコンパイラ，リンカーが存在しない状況を設定し、sample.forという名前のソースのコンパイル，リンク実行を行った。

## 5．　むすび

本稿では、NOS の論理構造を明確にし、開発及び保守を容易にする系統的な設計法について考察した。前半では、設計の基本概念について述べた。具体的には、まずネットワークを基本とした情報処理システムを仮想機械(VM)を用いてモデル化した。次に、このVMを機能単位に順次、段階的に分解し適当な分解レベルで、実システムとの対応をとり、分解されたVMの記述とVM間の関係を記述することによりNOS を含むシステムの概論設計を与えた。

後半では、前半で導入した設計基本概念に基いて、ここでは、パソコンを用いたNOS の構成に限定し、論理設計を行った。VMの仕様記述には、オートマトン表現を用い規定した。これをもとに、CP／M の機能を用いながらPC8001とIF800 上にNOS の基本的な機能を実現した。最後に、実験結果を整理し要約した。

今後の課題としては、前半で導入した基本概念をマルチタスキング環境下のNOS の構成に適応することであり、現在、オブジェクト思考の概念もとり込んで開発中である。

参考文献


[1] Randolph T.Yeh and K.Mani Chandy,"On the Design of Elementary Distributed Systems",Computer Networks,3(1979)

[2] Donnelly.J"Components of a Network Operating System",Computer Networks,3(1979)

[3] Richard W.Watson and John G.Fletcher"An Architecture for Support of Network Operating System Services",Computer Networks,4(1980)

[4] A.D.Birrell and R.M.Needham,"A Universal Fike Server",IE$^3$trans.on S.E. SE-6,No.5(1980)

[5] Marc Guillemont,"The CHORUS distributed operating system:Design and Implementation",Local Computer Networks.IFIP.1982

[6] Duen-Ping Tsay and Ming T.LIU,"MIKE:A Network Operating System for the Distributed Double-Loop Computer Network",IE$^3$ trans.on SE,SE-9 No.2(1983)

[7] 野口,元岡"オペレーティングシステムの記述に関する一考察"情報処理,vol.14,no.2(1973)

[8] N.Saito,J.Murai,O.Nakamura,"Design Principle of Operating System for Local Area Network",情報処理学会,ローカルエリアネットワークシンポジウム(1983)

```
A>nos                                   ← CP/Mコマンドレベルにおいて"NOS"
                                          とタイプすることによりNOSが
                                          起動されCIがプロンプト"NOS>"
                                          を出す
NOS>type dskifa:sin.for crtpc:          ← DISKIFA上のファイルSIN.FORを
                                          CRTPCへ印字するTYPEコマンド
      integer st,ed,stp
      logical gr(51)
      pi=3.14159/180.0
      call open(6,'SIN      DAT',0)
      read(6,200) st,ed,stp
  200 format(3i5)
      do 10 i=st,ed,stp
      do 20 j=1,51
   20 gr(j)=' '
      x=float(i-1)
      si=sin(x*pi)
      ci=cos(x*pi)
      is=si*25.0+25.5                   ← ファイルの内容
      ic=ci*25.0+25.5
      gr(is)=1hS
      gr(ic)=1hC
      write(1,110) x,si,ci,(gr(j),j=1,51)
  110 format(1h ,3f5.1,'    ',51a1)
   10 continue
      stop
      end

NOS>exec dskpcb:sample.for fort06.dat   ← DISKPCB上のFortranソース
                                          プログラムSAMPLE.FORをどこ
                                          にあるかわからないFORT06.
                                          DATファイルを入力データとして
                                          実行するEXECコマンド
[Compile]
SHAIN                                   ← EXEC APが出したメッセージ
[Link]                                  ← コンパイラが出したメッセージ

Data     0103      1944                 ← リンカが出したメッセージ

[0127    1944        25]
[Begin execution]

          10         23       230       529
         125         63      7875      3969
         145         12      1740       144
          96         24      2304       576
          85         39      3315      1521   ← 実行結果
          41         28      1148       784
          98        143     14014     20449
           5         96       480      9216
          12         75       900      5625 STOP

NOS>←再びCIへ戻りネットワークコマンドを待つ
```

図４．３　ＴＹＰＥとＥＸＥＣコマンドの実行例