資　料

ミニコンピュータ複合体とそのオペレーティング・システム*

松浦敏雄** 酒元登志克** 矢野秀一郎**
藤井　護*** 都倉信樹** 岡本卓爾****

Abstract

A resource sharing minicomputer complex system has been designed and implemented, which consists of three PDP-11 systems, a data path called Common Bus and three microoprocessor-controlled interface processors connecting PDP-II systems to Common Bus. An perating system for this complex system was implemented by adding the following functions to a single user disk operating system (DOS) supplied by the manufacturer, (i) DOS-to-DOS communication facilities, (ii) the function to service multiple requests from other systems and so on.

Some of the features of this system are as follows: (1) slight modifications of the DOS were required, because the interface processors provide powerful process-to-process communication facilities; (2) a user on a PDP-11 system can make use of all resources in this complex system as if they were of it own.

1. まえがき

ミニコンピュータ複合体については，これまでいくつかの試みがなされている．これらは，大型機なみの性能を持たせることを目的としたものと，各種の資源を共有して各ミニコン・システムの持つ長所を相互に享受し，かつその弱点を相互に補完することを目的としたものとに大別できよう[1),2)]．筆者らは先に後者の目的で既製のインタフェイス装置を用いた複合体のオペレーティング・システムを開発した[3)]．今回，これとは全く独立に，実験研究用のシステムとしてより拡張性に富み柔軟なリソース・シェアリング・システムを，結合方式やインタフェイス装置の設計・製作を含めて開発したのでここに報告する．本システムは，機器構成上の相異が比較的顕著な既設の PDP-11，3セットを対象としている．

ハードウェアの面では，マイクロプロセッサ*****を内蔵して融通性に富んだインタフェイス装置を介して共通バスに結合する方式を採用し，ミニコンの増設にも容易に応じられる等，拡張性の高いものとなっている．

ソフトウェアの面では，使い勝手が良く利用者の多い単一ユーザ用の既存のオペレーティング・システム (DOS) を改造した OS を開発した．これは外部仕様としては DOS を完全に含み，他のミニコンの持つリソースをあたかも自システムが持っているかのようにプログラミングできるという意味で使い易いリソース・シェアリングの機能を持ったものである．

2. ハードウェア

2.1 設計方針

本複合体の結合方式は，実験研究用のシステムとして機能面で柔軟性に富み，かつ各部の動作状況を把握し易いことが望ましい．また，既存のミニコンの従来の使用法や開発のコストも十分考慮しなければならない．ここでは，このような制約のもとで，次のような

* A Minicomputer Complex and Its Operating System by Toshio MATSUURA, Toshikatsu SAKEMOTO, Shuichiro YANO, Nobuki TOKURA (Faculty of Engineering Science, Osaka University). Mamoru FUJII (Computation Center, Osaka University) and Takuji OKAMOTO (Faculty of Engineering, Okayama University)
** 大阪大学基礎工学部情報工学科
*** 大阪大学大型計算機センター
**** 岡山大学工学部電子工学科
***** マイクロプロセッサとしては，設計当初に望み得た最良の Intel 8080 を用いている．

基本方針をとった.

1) 同機種のミニコンを用いる利点を生かし，システム全体にわたって一様性を図ることにより，経済的で拡張性に富んだ結合方式とすること.

2) ミニコンの負担をできるだけ少なくすること.

3) ミニコンの増設が容易なこと.

4) ミニコンのハードウェアは改造しないこと.

5) 必要に応じてミニコンを切り離せること.

以上の設計方針に基づき，既存のミニコン3システムをミニコンのバス（ユニバス）とほぼ同じ構造の共通バスで接続し，各システムと共通バスとのインタフェイスには安価なマイクロプロセッサを用いた.

本システムの特徴を列挙する.

1) 分散制御形の非同期式共通バスを用いて高速で簡単な結合方式を経済的に実現していること.

2) インタフェイスにはマイクロプロセッサを用いているのでミニコンのソフトウェアの負担を軽減でき，しかも，インタフェイス機能がソフト的に可変であることから，各種の方法を試験でき，実験研究用として有効であること.

3) ユニバスや共通バス上に，任意の目的のマイクロプロセッサを増設できること.

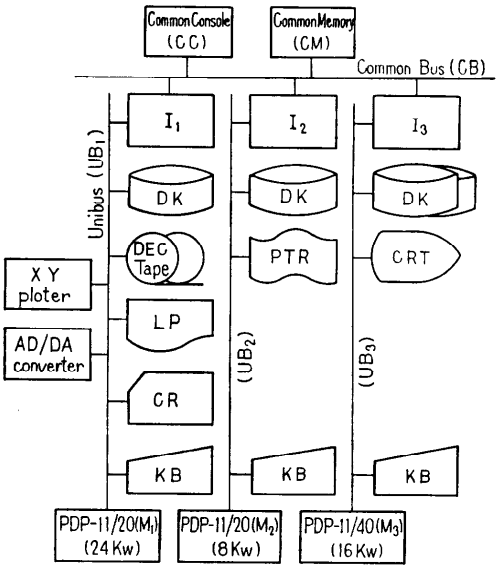
4) 以上のように，ハードウェア，ソフトウェアともに相当柔軟で，システムの拡張も容易であること.

2.2 システム構成

Fig. 1 にシステム全体の構成を示す. 各ミニコン(PDP-11)[4]M_i* ($i=1,2,3$) のユニバス UB_i は，同一構造のインタフェイス装置 I_i を介して共通バス CB に接続されている. I_i はマイクロプロセッサ (Intel 8080)[5] μ_i を内蔵している. I_i は M_i から見たとき，UB_i 上の単なる一つのデバイスとなっており，UB_i 上の他のデバイスと同様に，I_i は UB_i 上の任意のアドレス (M_i の主記憶の各セルやデバイスのレジスタ類) に直接アクセスできる.

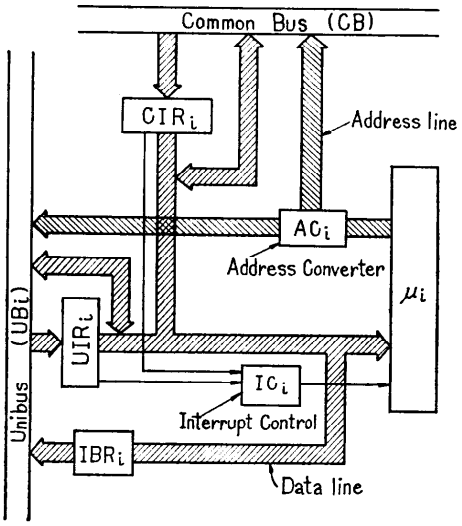
CB 上には，RAM を用いた共有メモリ CM と，システム全体の保守やデバッグのための共通コンソール CC が置かれているが，この他に任意の装置を接続することもできる. I_i は CB 上の任意のアドレスにもアクセスできる. CB の制御は，UB の制御とは異なり，アービタと呼ばれる機構で簡単かつ高速に分散制御している[6]. CB は，この機構を除いて UB と全く同じ構造である. これにより，I_i から見た UB_i と CB は基本的に同じとなり，I_i のハードウェア，ソフトウェア共に簡単化を図ることができる(**Fig. 2**). UB_i—CB 間のデータ転送はすべて μ_i の制御の下に行われる. μ_i の 16 ビットのアドレスは，アドレス変換機構 AC によって，UB_i か CB 上のアドレスに変換される (2.3 参照).

* 特に必要がなければ，以下，添字を省く.



I_i: μ_i (Intel 8080)を内蔵したインタフェイス

Fig. 1 Configuration of the complex system



CIR_i: Common Bus Interrupt Register
UIR_i: Unibus Interrupt Register
IBR_i: Interrupt Buffer Register.

Fig. 2 Structure of the interface I_i

μ_i から出力される8ビットのデータは，2回分を1語（16ビット）として組み立て，UB_i や CB に乗せることもできる．入力の場合も同様である．

M_i から μ_i への通信は割り込みによる．M_i が I_i にデータを送ると μ_i への割り込みが発生する．μ_j から μ_i $(i \neq j)$ への通信や，μ_i から M_i への通信も同様である（2.4 参照）．

CM には通信のための μ のプログラムやデータが格納される．このメモリには，競合防止のための1ビットのハードウェア・セマフォが設けられている．

I_i には，以上のほか Fig. 2 では省略してあるが，単体のデバッグやモニタリングに必要な機能を備えたコンソール μC_i がある．

2.3 アドレス変換機構

UB_i 及び CB 上のアドレス空間は 4k 語（2^{13} バイト）単位のページに分割されている．AC は8コの4ビット・レジスタを内蔵しており，μ_i から出力された16ビットのアドレスは，その上位3ビットがそれで指定されたアドレス変換レジスタの内容4ビットで置き換えられる．この17ビットに変換されたアドレスの最上位ビットは，残りの16ビットが UB_i 上のアドレスか CB 上のアドレスかを示す．

アドレス変換レジスタの内容は，μ_i が入出力命令を実行することによって読み書きできるほか，特定のレジスタはイニシャル・ロード時に μC_i から手動でセットすることができる．

2.4 割込み処理機構

2.4.1 M_i から μ_i への割り込み

M_i が I_i 内のレジスタ UIR_i にデータを書き込むと μ_i への割り込みが生ずる．このとき，前回の割り込みが μ_i によってまだ受け付けられていない場合には，UIR_i への二重書き込みとなるので，書き込む前に，UIR_i が空かどうかを示すステータス・ビットをソフトウェアで確めねばならない．URI_i にデータが入るとステータス・ビットはセットされ，CB 側からの割り込みとの競合を先着順で処理する割り込み制御機構 IC を経て μ_i に割り込み要求が発せられる．μ_i は割り込みを受けつけると，特定のアドレスから始まる割り込み処理ルーチンを実行し，UIR_i の内容を入力命令で読み込み（このとき，ステータス・ビットは自動的にリセットされる），それを解読して所定の処理を行う．

2.4.2 μ_j から μ_i への割り込み

μ_j から μ_i への割り込みも，基本的には 2.4.1 の場合と同様である．異なるのは，一般には複数個のマイクロプロセッサから μ_i への割り込み要求が同時的に発生する点である．これらの要求は，すべて同一のレジスタ CIR_i にデータを書き込むことによって満たされるが，2.4.1 の場合のように，CIR_i が空かどうかを各 μ_i がソフトウェアでテストするとすれば，その間，μ_j が CB の使用権を確保し続けなければならず．そのためのハードウェアが必要となり，時間のロスも大きい．そこで，本システムでは，CB からの CIR_i の使用は，ハードウェアで実現した“テスト・アンド・セット”の機構を用いている．

2.4.3 μ_i から M_i への割り込み

μ_i から M_i への割り込み機構は，M_i のそれをそのまま用いている．μ_i が出力命令でレジスタ IBR_i にデータを書き込むと，M_i に割り込み要求が発せられ，M_i がその割り込み処理ルーチンで IBR_i の内容を読み込み，解読して所定の処理を行う．μ_i が IBR_i にデータを二重書き込みしないように，ここにも“テスト・アンド・セット”機構を用いている．

3. マイクロプロセッサのソフトウェア

3.1 概　要

本章では，プロセス間通信のために開発した μ のソフトウェアについて述べる．μ のソフトウェアはすべてリエントラントルーチンで構成され，CM 内に格納されており，各 μ_i は独立にこれを実行している．

前述のように，μ を介した情報伝達のために，16ビットパラメータ（割り込みパラメータと呼ぶ）をもつ割り込み機構が設けられている．異なるサブシステム*内にあるプロセス間では，これを利用して情報伝達がなされる．M_i 内のプロセス P から M_j $(j \neq i)$ 内のプロセスQに対してデータを転送しようとするとき，M_i から μ_i，μ_i から μ_j，μ_j から M_j へ順次割り込みをかけ，目的のプロセス Q へ情報を伝達する．

情報を授受する2つのプロセス P, Q 間にはあらかじめルートが開設されていなければならない．ルートは全二重の通信路に対応するもので，P, Q は同じロジカルなルート名を指定して μ に要求することによって，フィジカルなルート番号が与えられる．以後の P, Q 間の通信はロジカルルート名だけで行える．

3.2 構　成

μ のソフトウェアは，割り込み処理ルーチン，メイ

---

* M_i と UB_i 上の I_i 以外のすべてのデバイスを含めてサブシステムと呼ぶ．

ンルーチンおよびリセット処理ルーチンの3つのルーチンから構成されている．メインルーチンは，ルート登録ルーチン，ブロック転送ルーチン，ルート閉鎖ルーチンなどのシステムサブルーチン (SSR と略す) を含んでいる．これらすべてのルーチンは，デッドロックを防ぐため他の μ や M からの割り込みを待たないように作製されており，自分自身で完結した仕事を行う．

割り込み処理ルーチンは，M_i および他の μ_j から μ_i への割り込みに際して動作し，原則として割り込みパラメータを μ_i の待行列につなぐ仕事を行う．メインルーチンは，待行列*から割り込みパラメータを取り出し，その内容に応じた SSR を実行する．リセット処理ルーチンは，主としてシステム起動時に初期化を行う．

3.3 情報の伝達

3.3.1 割り込みパラメータ

本通信方式で重要な役割を果たす割り込みパラメータの形式について説明する．割り込みパラメータには，**Fig. 3** で示すようにバイトモードとアドレスモードの2つの形式がある．これらは割り込みパラメータの第 15 ビットの値で区別される．

バイトモードは 16 ビットの割り込みパラメータ内にすべての情報が含まれており，このパラメータだけで，μは完結した処理を行う．このパラメータの第13～8ビットは，第 14 ビットが“1”のとき SSR の番号を，“0”のときルート番号をあらわす．第 7～0 ビットは SSR または相手のプロセスに渡すデータである．このように M_i 上のプロセスは μ_i の SSR を直接呼び出せるだけでなく，ルート番号とデータを与えるだけで相手のプロセスまでの転送が可能である．

アドレスモードは，第13～0ビットで示されるアドレスに，SSRの番号が書かれており，さらに続く何バイトかに，データが書かれている．このバイト数は SSRによって異なる．なお第 14 ビットのプライオリティは割り込み処理ルーチンで待行列につなぐときに用いられる．このモードでは，ルート番号を指定してデータを転送することはできない．

3.3.2 ルートの開設

互いに通信を行いたい2つのプロセス P_1, P_2 (それぞれ M_1, M_2 上で稼動しているものとする) は，同じルート名 (ロジカルルート名) でルート開設要求を出さなければならない．それは P_i が通信に必要な情報の書かれたテーブル (U テーブルと呼ぶ，**Fig. 4** (a) 参照) を準備し，μ_i にこのテーブルをデータとするアドレスモードで割り込むことによってなされる．この要求は P_1 と P_2 で非同期に発せられるが，以下，P_1 が先にルートの開設要求を出したものとする．

まず P_1 からの割り込みによって μ_1 はルート登録ルーチンを実行し，相手のプロセスのルート開設要求がすでに登録済であるかどうかを調べる．この場合まだ登録されてないので，CM 内にロジカルルート名でマッチングをとるためのランデブーテーブル[7],**の要素 (Cテーブルと呼ぶ) を作り，登録を完了する．次に，P_2 が要求を出したとき，今度は P_1 が登録したCテーブルがすでにできているので，このテーブル上

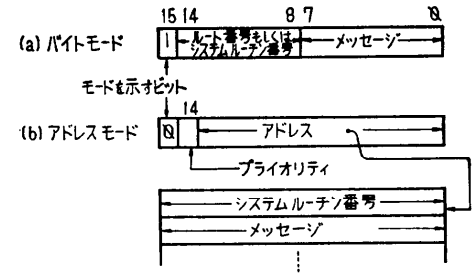


Fig. 3 Format of the interrupt parameter

システムルーチン番号*1*3
パラメータ*1
ルート番号*2
ルートステータス*2
ロジカルルート名*1
ユーザ・コード*1
通信相手のユーザコード*1

*1…ルート開設前に M_i がセットすべきところ
*2…ルートが開設されたときに μ_i がセットするところ
*3…ルート登録ルーチンのルーチン番号をセットする

システムルーチン番号*4*7
パラメータ
ルート番号
ルートステータス
送手側バッファポインタ*4
送手側バイト数*4
受手側バッファポインタ*5
受手側バッファサイズ*5
転送されてきたバイト数*6

*4…送り手側のときに M_i がセットすべきところ
*5…受け手側のときに M_i がセットすべきところ
*6…μ_i が受け手側に転送されたデータ数を知らせるところ
*7…ブロック転送ルーチンのルーチン番号をセットする

(a) ルート開設時　　(b) ブロック転送時

(注) この図では説明上不用と思われる項目は省略してある．

Fig. 4 Format of the U Table

* 他の μ_j あるいは M_i への割り込みの際に相手が割り込み禁止状態のときにもこの待行列にいったんつながれて，後で再び割り込みがなされる．これは，相手が割り込み禁止解除になるまで，待つことにすればデッドロックになるおそれがあるためである．

** ランデブーテーブルはリスト構造になっている．

でマッチングがとれる．続いてルート登録ルーチンは空いているルート番号を，Cテーブルおよび P_1 と P_2 のUテーブルに書き込み，ルートの開設を完了する．以後の P_1 と P_2 間の通信は，すべてこれらの3つのテーブルを介して行われる．

3.3.3 バイト転送の基本操作

ルート番号が与えられてからのプロセス間の通信は簡単である．以下に P_1 から P_2 へのバイト転送について説明する．P_1 はルート番号と転送したい1バイトのデータを1ワードにパックして，バイトモードで μ_1 に割り込みをかける．μ_1 はルート番号によって，C テーブルを見て割り込み相手を知り，μ_2 に割り込みをかける．μ_2 は割り込まれたままのパラメータ形式で M_2 に割り込みをかける．このようにしてプロセス間のバイト転送が実行される*．

3.3.4 ブロック転送の基本操作

すでにルートが開設されている2つのプロセス P_1 から P_2 へ n バイトのデータ転送を行う場合，P_1 は Fig. 4 (b) で示すように U テーブルをセットし，次にこの U テーブルをデータとするアドレスモードで μ_1 に割り込みをかける．μ_1 は U テーブル内のルーチン番号で示されるブロック転送ルーチンを起動し，CM 内にバッファを確保して，転送したいデータをコピーし，μ_2 に割り込んでブロック転送のルーチンを動かす．このルーチンは，P_2 の U テーブルを見て，サイズが n バイト以上のバッファが割り付けられているかを調べる．もしバッファが割り付けられていないとき，あるいはバッファのサイズが小さいときは，転送したいデータのバイト数をUテーブルに書き込み，M_2 に割り込む．すでにバッファが割り付けられているときは，そのバッファにデータを書き，さらに転送したデータのバイト数をUテーブルに書き込んで M_2 に割り込む．前者の状態で割り込まれた場合，M_2 は十分なサイズのバッファを割り付けて，再びデータの転送を依頼するために，μ_2 に割り込む．割り込まれた μ_2 はもう一度同じルーチンを起動してデータを転送する**．

3.3.5 ルートの閉鎖

P_1, P_2 間の通信が終って，どちらか一方のプロセス（P_1 とする）がルートを閉鎖しようとする場合について述べる．P_1 は μ_1 のルート閉鎖ルーチンを呼び出す．このルーチンは μ_2 を介して，P_2 の U テーブル内のルートが開設されていることを示すビットをたおす．さらに CM 内の対応する C テーブルを消去し，使用していたルート番号を解放する．以上でルートの閉鎖が完了する．

3.4 ユーザプログラム間のブロック転送

異なったサブシステム内にあるユーザプログラム間のブロック転送も，ルートを開設し，前述の μ のバイト転送，ブロック転送の機能を用いて，送り手と受け手の同期をとり，データを転送すればよい．しかしながらユーザが個々にこれらの手続きを細かく記述するのは面倒である．そこで，DOS の I/O マクロ（後述）と同じように，転送要求をマクロで書けるようにして，ブロック転送が簡単に行えるようにしている．

以下に各マクロについて説明する (**Fig. 5**)．.OPENP はルートの開設のためのマクロで，その引数はロジカルルート名等が書かれているルートブロックのアドレスで，以後ルートの識別のために用いられる．.SENDはデータを転送するためのマクロで，その第2引数はデータバッファの先頭アドレス，バッファサイズ，転送のモードとステータスが書かれているバッファヘッダのアドレスである．.RECV はデータを受け取るためのマクロで，その引数は .SEND と同じである．.WAITP は転送の完了を待つためのマクロで，転送が完了するまで以下の命令は実行されない．.CLSEP はルートを閉鎖するためのマクロである．

次に内部仕様について述べる．.SEND 要求と .RECV 要求の対応がついたとき転送が行われるのであるが，これに先だち，受け手はバッファサイズを送り手に知らせる．送り手はこの情報と，送り手側のバイト数とで，転送すべきデータ数を決定し，μ のブロック転送ルーチンを用いてデータを転送する．ここで，送

```
(送り手側)                          (受け手側)
.OPENP  #RUTBLK                     .OPENP  #RTBK
.SEND   #RUTBLK, #BUFHDR            .RECV   #RTBK, #BUF
.WAITP  #RUTBLK                     .WAITP  #RTBK
.CLSEP  #RUTBLK                     .CLSEP  #RTBK

RUTBLK:(ロジカルルート名)           RTBK:(ロジカルルート名)
       (ユーザコード)                      (ユーザコード)
       (システム用)                        (システム用)
BUFHDR:(バイト数)                   BUF:(バイト数)
       (転送モード/ステータス)              (転送モード/ステータス)
       (バッファアドレス)                  (バッファアドレス)
```

Fig. 5 Procedures of block transfer

---

* 転送が終って，データを受け取ったことを送り手側のプロセスに知らせるには，返答ルーチンと呼ぶ SSR を呼び出せばよい．これによって，送り手側のUテーブル内の転送終了を示すステータス・ビットがセットされる．またこのとき送り手側のプロセスに割り込むか否かはパラメータで指定できる．

** 前の脚注と同じ．

り手側と受け手側のバッファサイズが異なった場合に，どのように対処するかを述べる．受け手側のバッファサイズの方が大きい場合には問題は生じないが，その逆の場合の対策として，ユーザは転送モードを指定することによって，次の2つの方法のうちいずれか一方を選択できる．

1) 受け手側のバッファサイズで転送を打ち切る．このとき送り手側は転送が完了したものとみなして，.WAITP をぬける．

2) 受け手側のバッファをいっぱいにしたあと，送り手側は転送中の状態で .WAITP で待つ*．受け手側は制御をユーザに返し，ここでユーザが再び .RECV を行うと，先ほどの続きのデータが転送される．データが全部転送されると送り手側は .WAITP を脱出する．

1) の方法は，データの最大長がわかっている場合，あるいは比較的少量のデータの転送に適しており，2) の方法は，大量のデータを転送し，逐次処理をするのに適している．

4. 複合体のオペレーティング・システム

既に述べたハードウェアと情報伝達機能をもとにしてリソース・シェアリングを目的とした複合体の OS を作製した．複合体全体の管理は，集中的には行わず，それぞれのサブシステムの OS が行うという分散制御の方式により実現している．

4.1 設計方針

OS はメーカ提供の DOS[8),9)]を基盤として下記の設計方針で作製した．

1) 各サブシステムにいるユーザ各々が，CPU，主記憶，コンソールを除く全システムのリソースを，他のユーザが使用中かどうかにかかわらず従来の DOS と全く同一の手続きで使用できること．

2) DOS の下で動作する各種のユーティリティ・プログラムや今までに開発されたプログラムは修正しなくとも実行できること．

3) 低速デバイスに対してはスプーリング処理を行い，効率の向上を図ること．

4) DOS の内部仕様の変更を極力少なくすること．

5) 新しくサブシステムが追加されてもソフトウェアは一切変更しなくてもよいこと．

4.2 方式の検討

DOS は，ユーザがコンソールを通じてファイル管理プログラム，コンパイラ，実行制御ルーチンなどと通信を行いながら，インタラクティブにプログラムの作製，実行を行うのに適した OS であり，ユーザが I/O マクロを発することによりデバイスを利用することができるようになっている．I/O マクロはモニタ内の I/O マクロ解析ルーチンで解析された後，各々のロジカル I/O ルーチン (LIO) で処理され，さらに必要ならばフィジカル I/O ルーチン (PIO) の処理を受ける．LIO には，ファイルの登録や管理，オープン中のファイルをロックする作業，二次記憶装置における記憶領域の使用状態を示すビットマップの管理等を行うものがある．PIO は各デバイスに対して用意されており，LIO からの要求により，デバイスの制御装置を直接操作し，データの転送，割り込みの処理を行う．

このような DOS の I/O 処理機能を基盤としてリソース・シェアリングを行うには，これらの I/O マクロをいかに処理するかが問題となる．これには次の2つの方法が考えられる**．

① PIO レベルで処理を行う方法：他のサブシステムに属すデバイスに対しても自システム内に PIO (疑似 PIO) を用意し，この疑似 PIO がそのデバイスを持つサブシステムの PIO と通信を行って目的の処理を達する．

② LIO レベルで処理を行う方法：ユーザが他のサブシステムに属するデバイスに対する I/O マクロを発すると，OS は相手の OS にそれを送信する．相手の OS はその I/O マクロの処理を行い，結果を返す．

以上の方法を比較すると，①の方法では，ファイルやビットマップの管理は LIO が行うので，1つのファイルや1つのデバイスのビットマップの管理を同時に多数のサブシステム内にある LIO が行うことになり，それらの間で非常に多くの通信を行う必要がある．また，デバイスが追加されるごとに，疑似 PIO を作製しなければならない．② の方法では，① の方法に比べ，I/O 処理依頼を受けた側の OS のオーバヘッドは大きいが，ファイルやビットマップの管理は単一の LIO のみが行うので通信の回数は少なくてすむ．また，デバイスが新しく追加されても手を加える必要はない．以上の理由から ② の方法を採用することと

---

* もちろんこのとき他に仕事があるときはその仕事をしながら待つこともできる (DOS の .WAITR の機能)．

** このほか，ユーザが他のサブシステムにあるファイルを使うに先だってそのファイルを自サブシステムの二次記憶装置にコピーする方法もあるが，これは二次記憶装置の使用効率が低下するだけでなく，ファイル管理プログラムの大幅な変更を必要とする．

した．

4.3 外部仕様

本システムを利用するユーザは空いているコンソールを探し，従来と同じようにログイン操作を行う．以後はすべてのリソースが自分の確保したサブシステムに属しているものとみなして以下の点を除き従来のDOSと全く同じ手続きでシステムを利用できる．他サブシステムに属するデバイスを使用する場合，従来のデバイス名にサブシステム名を表わす1文字を付け加えた3文字のデバイス名で指定しなければならない．

4.4 内部仕様

DOSの主な改造点及びエラー処理について述べる．

4.4.1 デバイス・ドライバ・リスト (DDL)

従来のDOSはシステム生成時に自システムに属するデバイスについて，そのPIOのサイズや開始番地等が記入されたリスト (DDL) を完成する．これらのDDLだけでは他のサブシステムに属するデバイスについての情報は与えられないので，複合体のOSにはそれらの情報(そのデバイスが属するサブシステム名)をもったDDLを生成する機能が付け加えられた．

4.4.2 I/O マクロ解析ルーチン

他のサブシステムに属するデバイスに対するI/Oマクロは，自サブシステム内だけでは処理できない．そこで，このようなI/Oマクロの処理は指定されたデバイスをもつサブシステムに依頼する必要がある．このために，I/Oマクロ解析ルーチンに以下の機能が付け加えられた．

DDLを調べ，I/Oマクロで指定されたデバイスが他のサブシステムに属すなら，それを管理するOSに対し，OS間ルート(後述)を使ってI/Oマクロ処理の依頼を発し，必要ならばデータの転送も行う．

4.4.3 OS 間の通信

下記の用件に関しOS間で通信を行う必要がある．

1) I/Oマクロの処理を依頼する．
2) I/Oマクロの処理の結果を返す．
3) 依頼主にエラー・メッセージを伝える．
4) 依頼したI/Oマクロ処理に関するコンソールからの指示（中止や再開など）を伝える．

これらの通信は **3.3** に述べた μ のソフトウェアの助けをかりて行われる．

OS間で通信を行う場合，そのOS間にルートが開設されていなければならない．各 M_i のOSはシステム生成時に各 M_j $(i \neq j)$ のOSとのルートを開設するように μ_i に要求を出す．このような要求を出したOSのすべての組み合わせに対してそれぞれルートが開設される．M_i のOSは M_j のOSとのルートが開設されれば，M_j のリソースを共有できると判断する．

プロセスが出した要求は，16語のコントロール・ブロックにパックされた後，ブロック転送の待ち行列につながれ，受け手のOSに転送される．また，I/O動作に伴うデータの転送もOS間ルートを用いて行われる．コントロール・ブロックやデータを受け取ったOSは，バイト転送によって受け取ったことを送り手に伝える[10]．これらの動作は各OS間ルートで独立に行われる．

4.4.4 I/O マクロの管理

従来のDOSにはマルチプログラミングの機能が無く，複数のプロセスからの要求を処理することが考慮されていない．そこで要求のあったI/Oマクロを待ち行列につなぎ，これを管理する機能が付け加えられた．

4.4.5 スプーリング処理

低速デバイスに対しては，スプーリング処理を行い，あたかも多数のユーザが同時に自分が占有しているかのごとく使えるようにした．例えば，ラインプリンタでは，そのPIOを改造し，出力データはすべて二次記憶装置内に格納し，ラインプリンタが空いたとき，格納されているデータを出力する．

4.4.6 エラー処理

他のサブシステムから依頼されたI/O処理の実行中に起こりうるエラーの種類は大きく分けて2つである．それぞれのエラーの特徴及び処理について述べる．

1) フェイタル・エラー：これは，必要な領域が確保できなかったとか，システムにないデバイスへの要求を出したときなどに生じ，致命的なエラーであり，回復は不可能である．この種のエラーが生じたとき，OSは依頼主にエラー・メッセージを送り，待ち行列内の次の要求を処理する．エラーが生じた要求は放棄される．
2) アクション・エラー：使用するデバイスがオフラインになっていたときなどに生じ，ある動作(デバイスをオンラインにする等)を行った後，処理の続行が可能なものである．この種のエラーが生じたとき，OSは依頼主にエラー・メッセージを送り，依頼主から処理再開の要求が来れば続行できるようにエラー発生時の状態を記憶しておいて，次の要求の処理に移る．

サブシステム間のブロック転送に関しては，受け手側がチェックサムでエラーの検出を行い，エラーが発生したときは，送り手側にバイト転送で知らせる．

5. むすび

マイクロプロセッサのソフトウェアの開発は PDP-11 上でのクロスアセンブラおよび PDP-11 を用いたデバッガを作製して行った．またハードウェアのテストのための PDP-11 とマイクロプロセッサのルーチンを作製した．複合体のソフトウェアの開発に際して困難であった点は DOS の解読に相当量の時間を要したことと，マイクロプロセッサのレジスタが少ないためにリエントラントルーチンの作製が容易でなかったことがあげられる．

マイクロプロセッサの OS のサイズは 3k バイトであり，ミニコンの OS は DOS に比べて 2k バイト増加し，71kバイトとなり，常駐部は 1kバイト増加して 4.8kバイトとなった．また他サブシステムに子タスクを発生させる機能を設計製作中である．これにより，システムの有用性が更に向上すると思われる．

日頃御指導頂く大阪大学嵩忠雄教授，熱心に御討論頂いた嵩研究室の諸氏および御協力頂いた岡山大学電子回路研究室の諸氏に深く感謝します．とくに，谷口健一，故細見輝政，奥井 順，葛山善基，本藤勉，前村義明の諸氏に厚く感謝します．

参考文献

1) 小特集：コンピュータ・コンプレックス，情報処理，Vol. 15, No. 7, pp. 524～564 (1974).
2) 徳田他：KOCOS のアーキテクチャ，情報処理学会計算機アーキテクチャ研究会資料(1975).
3) 本田他：PDP-11/20 デュアルシステムにおけるデバイスシェアリングシステムの製作，情報処理，Vol. 14, No. 10, pp. 794～801 (1973).
4) PDP-11 Processor Handbook, Digital Equipment Corp. (1971).
5) Programming Manual for the 8080 Microcomputer System, パネトロン (1974).
6) 岡本他：非同期式リングアービタの一方式，信学論 D, Vol. J 59-D, No. 8, pp. 582～583 (1976).
7) D. C. Walden: A System for Inter-process Communication in a Resource Sharing Computer Network, Comm. ACM, Vol. 15, No. 4, pp. 221～230 (1972).
8) PDP-11 DOS/BATCH Software Support Manual, Digital Equipment Corp. (1974).
9) PDP-11 Disk Operating System Monitor Programmer's Hand book, Digital Equipment Corp. (1972).
10) D. L. Russell & T. H. Bredt: Error Resynchronization in Producer-Consumer Systems, ACM Operating Systems Review, Vol. 9, No. 5, pp. 106～113 (1975).

(昭和51年9月17日受付)