KOCOSのアーキテクチャ(2)
—「プロセス間通信方式とソフトウェア構成」

徳田英幸　上林憲行　竹山明　石塚朝生（慶大工学部）
西垣秀樹　平塚良治（沖電気工業(株)）

1. はじめに

近年，いろいろな分野において数多くのミニコンピュータ・コンプレックスが，提案され，又，稼動しはじめている。[1] しかし，それらの多くの場合，計算機間での結合方式に関しての研究に重点が置かれ，コンプレックスシステムの環境を生かしたオペレーティングシステムに関する研究は十分な成果をあげていないように思われる。オペレーティングシステムの作成に際しては，できるだけ既存のシステムソフトウェアを効率よく活用して，構成していく例が多く見られ，システム作成当初から，新しい概念で構成されたという例は少ない。　この主な理由としては，SUEシステム[2]などのようなコンピュータ・コンプレックスを意識したアーキテクチャをもつ計算機システムがあまり開発されておらず，従来からの単体のプロセッサを接続し，既存のソフトウェアに極力手をかけずに実現しようという要求が強かったためであろう。従って，既存のコンピュータ・コンプレックスには，相手プロセッサへそのプロセッサのリモート・ジョブ・エントリー機能を介して交信するといった"ルーズ"な結合形態が多く見られた。

汎用性のあるリソースシェアリング，ロードシェアリングを目的とし，さらに並列処理をめざしたミニコンピュータ・コンプレックスにおけるオペレーティングシステムに課せられた課題としては次のものがある。

[1] システム全体の信頼性の向上
[2] 各ミニコンピュータの処理能力の限界
[3] プロセス間通信におけるオーバーヘッドの軽減
[4] 使いやすさ，ユーザの生産性の向上
[5] システムの拡張性，可変性の実現
[6] システムソフトウェアの生産性の向上

筆者らは，これらの[1]~[6]を考慮し，ミニコンピュータコンプレックスの可能性を追求すべく，リソースシェアリング，ロードシェアリング，並列処理を指向する異種ミニコンピュータ・コンプレックス"KOCOS"（Keio-Oki's COmplex System）におけるオペレーティングシステム"KOCOS"（Keio-Oki's Complex Operating System）を設計した。本稿では，KOCOSの基本的性格であるプロセス間通信機能，システムソフトウェアの構成について述べる。

2. オペレーティングシステムKOCOSの基本的性格

KOCOSの機能は，単一バス方式による結合形態，接続しているミニコンピュータの処理能力，BIU（Bus Interface Unit）の中枢部に採用したマイクロプロセッサの機能，DBC（Distributed Bus Controller）やBM（Bus Monitor）の機能[3]と密接に関係しており，次のような性格をもっている。

[1] システムの管理機構は，"集中分散管理方式"を採用している。これは，システ

ム内に分散して存在するハードウェアリソースやソフトウェアリソースの管理，ジョブのスケジューリングに対しては，ＳＳ（System Scheduler）による集中管理方式をとり，各ＥＰ（Element Processor）内で実行されるユーザプロセスやシステムプロセスの実行時の制御に関しては，各ＥＰ上に実現されているＬＯＳ（Local Operating System）による分散管理方式をとることを示している。

[2] システムモジュールの一部に生じた障害が，システム全体をダウンさせるという事態を生じさせないよう，多レベルの障害対策機構を備え，システム全体の高信頼性を実現している。

[3] 従来のネットワークで採用されているメッセージ転送を主としたルーズなプロセス間通信機能ではなく，プロセス間での直接的な通信を実現している。さらに，プロセスを制御する同期操作も備えているので，コ・オペレーティブなプロセスの制御[4]が，容易に実現でき，並行プログラムを効率よく処理することができる。

2.1. プロセス間通信機能（ＩＰＣＦ：Interprocess Communication Facility）

プロセス間通信機能は，ミニコンピュータ・コンプレックスに課せられているリソースシェアリング，ロードシェアリング，並列処理環境の実現に関して，もっとも大きな要因であるといわれている。[5][6][7]

プロセス間でのメッセージ転送に際し，メッセージの組み立て，分解や送受信時のコントロール情報の解析といったわずらわしい作業がシステムオーバヘッドとして影響してくると，システムのスループットが低下するだけでなく，多量データを高速転送する必要のあるリアルタイム処理などは，著しく制約されてしまう。

従って，ミニコンピュータ・コンプレックスにおけるプロセス間通信機能としては，次のような性格を備えておくべきであろう。

[1] プロセス間通信の際のオーバヘッドを極力のぞき，各プロセッサへの負荷を軽減する。
[2] ユーザが，ある入出力装置に対して読み書きするのと同様に，プロセス間での交信が直接的にでき，かつ，スムーズに複数のプロセッサ内のプロセスによるコ・オペレーションが行える。
[3] メッセージ転送の際のエラーに対する自動再試行機構や，フォールバック機能を備えた高信頼性を保証する。
[4] メッセージ転送に関し，ブロック長やバス使用優先レベルなどの物理的転送パラメタもできるだけシステムのトラフィックにつりあうよう動的にセットでき，システム内で調和のとれたプロセス間通信ができる。
[5] ブロードキャスト転送や，通信をしている各プロセスが所有しているバッファ量の違いを吸収できる部分転送機能や，転送完了情報の遅延機能を備えている。

ＫＯＣＯＳでは，以上の点を考慮し，論理的な結合を密とするため，送信プロセスの番地空間から受信プロセスへの番地空間へ向けてのＤＭＡ（Direct Memory

Access）機能を生かした直接的な通信方法を採用した。さらに，メッセージ転送の際の種々の仕事や繁雑な割込み処理などから，プロセッサを解放し，ＢＩＵの中枢部に高度な知的管理機能を付加したので，オーバヘッドをなくし，ＬＯＳへの負荷を軽減しただけでなく，成長可能なＩＰＣＦを設計することができた。

このようなＩＰＣＦの階層構造は図-1.のように表わすことができる。

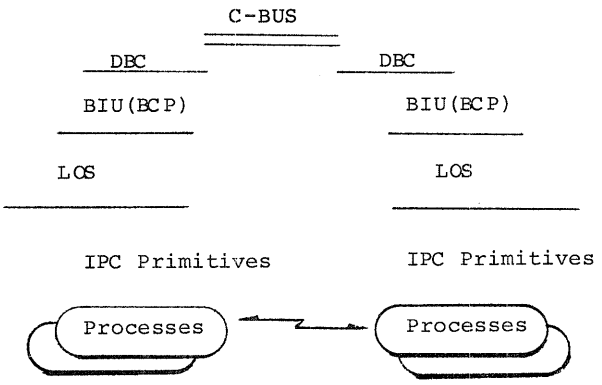


図-1. IPCFの階層構造

2.1.1. プロセス間通信基本命令（ＩＰＣ Primitives）

プロセス間通信は，プロセス間通信基本命令を実行することにより行うことができる。この基本命令は，すべてシステムコールという形式で実現されており，図-2.に示されるように分類することができる。

IPC Primitive	Message Transfer	Message Type	WRITE	READ
			BWRITE(Broadcast Write)	
		Force Type	FWRITE(Force Write)	WANSWER(Write Answer)
			FBWRITE(Force Write)	
			FREAD	RANSWER(Read Answer)
	Process Cooperation			WAITFIM (Wait Final Message)
		Control Type	START	STOP
			SIGNAL	WAIT
			RSTATUS1	
			RSTATUS2 (Return Status)	

図-2. プロセス間通信基本命令

メッセージ転送を行なう基本命令は，相手プロセスとのマッチングがメッセージ単位に，ＢＩＵのランデブーテーブル（後述）上で行なわれロジカルリンクが確立するメッセージタイプ（M-タイプ）と，相手プロセスに対し，強制的にメッセージ転送の旨を伝え，ロジカルリンクを確立するフォース・タイプ（F-タイプ）の２種類がある。プロセスの相互制御を行なう基本命令は，コントロール・タイプ（C-タイプ）といい，ＬＯＳの提供しているダイナミックなプロセスの制御機能を活用し，複数ＥＰ内のユーザプロセスやシステムプロセス間でのコ・オペレーションを実現している。

2.1.2. プロセス間通信方式

プロセス間通信方式については，プロセス間通信基本命令の３つのタイプに代表される次のような例をあげることができる。

(1) M-タイプ： あるプロセスαが，他ＥＰ上のプロセスβに対し，aa番地

以降n語をWRITEする場合。

(2) F-タイプ： あるプロセスαが，他ＥＰ上のプロセスβに対し，aa番地以降n語をFWRITE する場合。

(3) C-タイプ： あるプロセスαが，他ＥＰ上のプロセスβをwake upする場合。

これら(1)～(3)に対する交信手順は，それぞれ図-3.，図-4.，図-5.のように表わすことができる。ここでは，図-3.のM-タイプにおけるWRITE-READ について図中の番号にそって詳細について述べる。

1. プロセスαがシステムコールWRITEを実行する。
2. これを受けつけて，ＬＯＳ１のＩＰＣ handler（Interprocess Communication Handler）が，ＢＣＷ（BIU Control Word；チャネルコマンドを機能拡張したものに相当する。）の語数と，バス使用要求の優先順位の情報を知らせるためにＢＩＵ１に対してListem I/o と呼ぶ入出力命令を出す。次に，ＢＣＷのアドレスxxを知らせ，ＢＩＵ１に対して起動をかけるStart I/o 命令を実行する。この後，ＬＯＳ１は，プロセスαをwaiting状態にし，他プロセスにプロセッサを割り当てるためのスケジューリングを行う。
3. ＢＩＵ１は，ＢＣＷをＥＰ１のメインメモリから読み出し，その解読を行う。解読後，ファンクション部がWRITEであるので，ランデブーテーブル（ＲＴ；プロセス間でのメッセージ転送に関するコントロール情報のマッチングをとるためのデータポートをいう。）上で，プロセスα－β間の転送のためのパラメタをセットする。（但し，既に，プロセスβからのREAD要求が到着していればステップ５へ移る。）

4 プロセスβからのREAD要求が到着するまで，ＲＴ上で，待っている。

1' プロセスβが システムコールREADを実行する。

2' ＬＯＳ２は，これを受けつけ，Listem I/o，Start I/o 命令を実行する。

3' ＢＩＵ２は，Start I/o で知らされたＢＣＷのアドレスyyより読み込み，ファンクション部を解読後，READ であるので，yy番地以降6語をIPCM（Interprocess Communication Message；メッセージ転送のためのコントロール情報をいう。）として，ＢＩＵ１へ転送する。

4' ＢＩＵ１は，このＩＰＣＭを受信し，解読後，ＲＴ上でプロセスαとプロセスβとのマッチングが成立する。

5. ＢＩＵ１のＲＴ上でのマッチング成立後，ＢＩＵ１は，ＤＭＡを起動してプロセスαのaa番地以降n語をプロセスβのbb番地以降n語に向けて転送を開始する。転送完了後，ＢＩＵ１のＲＴ上のパラメタは解消され，転送終了をしめすＦＩＭ（Fimal Message）をプロセスα，βへ向けて転送する。

以上のようなプロセス間通信方式により，各ＬＯＳは，ＢＩＵに対してListen I/o，Start I/o命令を出すだけで，後のメッセージ転送に関しては，一切関与していない。転送に際して，ＲＴは必ず送信側ＢＩＵにセットされており，実際の転送管理を送信側ＢＩＵが行っている。このように，プロセス間通信の際の種々の仕事をＢＩＵへ負わせたので，以下のような特徴をもったＩＰＣＦを容易に実現することができた。

[1] プロセスの物理的位置に関係なく自由に，直接的にプロセス間通信が行える。すなわち，ＲＴ上でＩＰＣＭのマッチング成立後，送信プロセスの番地空間から受信プロセスの番地空間へＤＭＡ機能を利用してＬＯＳに全く無関係に転送を行う。従って，ＬＯＳは，メッセージ転送のための特別なバッファを

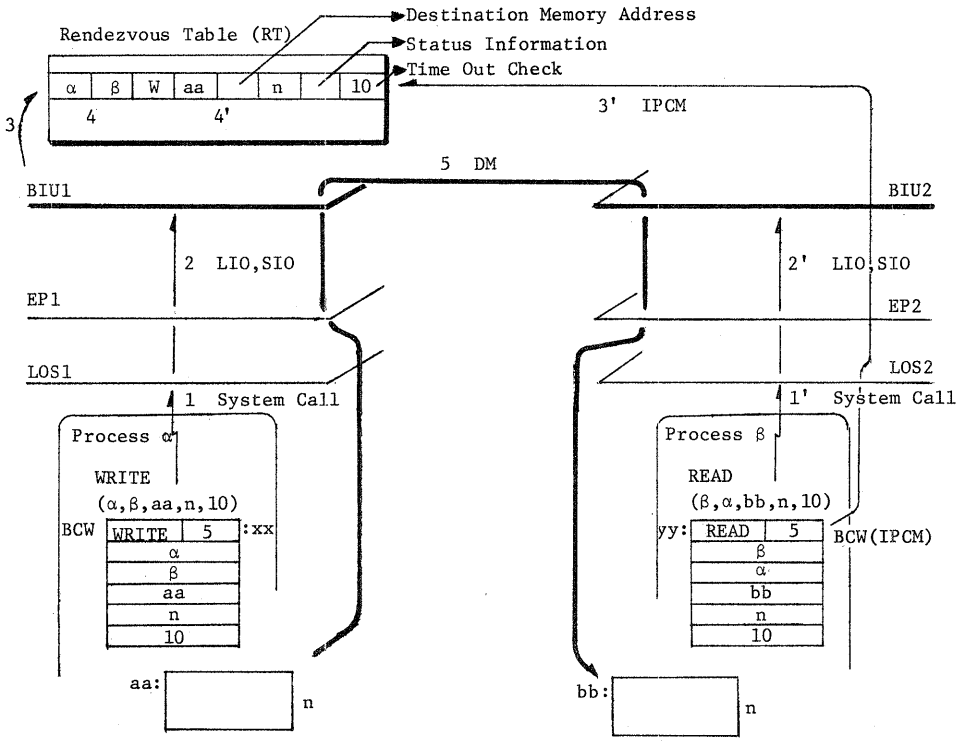


図-3. M-タイプ プロセス間通信例 (WRITE-READ)

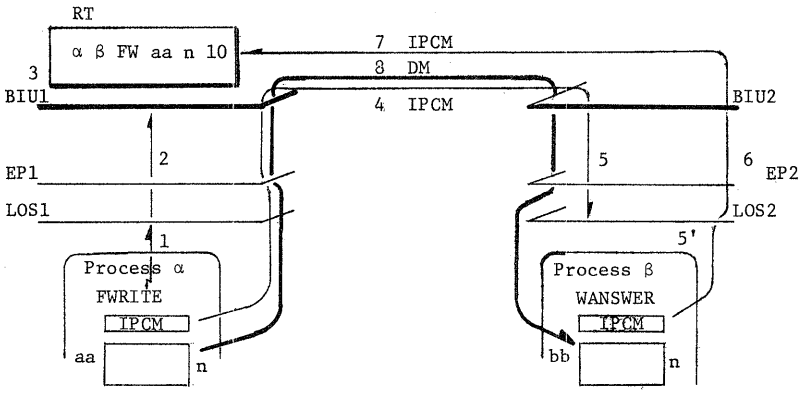


図-4. F-タイプ プロセス間通信例 (FWRITE-WANSWER)

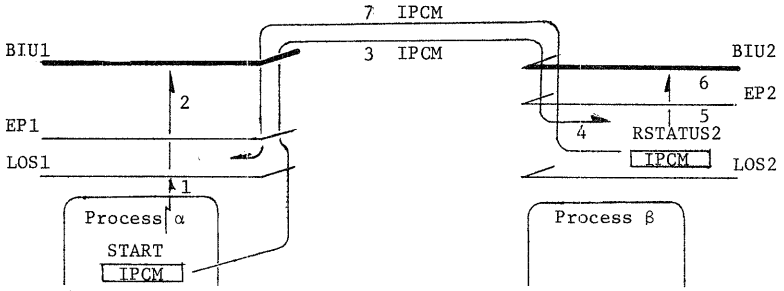


図-5. C-タイプ プロセス間通信例 (START-RSTATUS2)

持つ必要がなく負荷が軽減された。

[2] M-タイプ，F-タイプの２種類の転送方式が可能であり，部分メッセージ転送，離散型データ転送，one to any 転送の高度な機能を備えている。

[3] ＬＯＳの管理下にある複数プロセスによる同時送信，同時受信が可能であり，又，送信時には，メッセージの優先順位に基づいて送信可能である。

[4] ＩＰＣＭを基礎とした制御情報を介しての交信なので，結合形態に拘束されずに，機能拡張が容易に行える。

[5] Ｃタイプのプロセス間基本命令により，コンプレックス・システムの環境を生かしたコ・オペレーティング・プロセスが実現できる。

[6] ＬＯＳのサポートを受けず，ＩＰＣＦとして転送時のエラーに対するバックアップ機能を備え，高信頼性を実現している。

2.2 システム・ソフトウェア

2.2.1. システム管理形態――ＳＳとＬＯＳのロジカルな関係

ＫＯＣＯＳのようなコンピュータ・コンプレックスでは，ジョブの制御，リソースの管理を集中管理にするか分散管理にするかは重要な問題である。ＫＯＣＯＳではこの問題に対して，次のように考えて設計した。すなわち，これらの機能を各ＥＰに分散すると，システム全体の信頼性，負荷の分散化という面で望ましい。しかし，各ＥＰの処理能力にかなりの制約があるため，かえって負荷が増加するだけでなく，システムの拡張に不都合が生じてしまう。従って，これらを避けるため１台のＥＰにジョブの制御，リソースの管理を行わせる集中管理方式（この制御プログラムを実行するＥＰをシステム・スケジューラ（ＳＳ）という。）を採用した。ＳＳがダウンしたときには，他ＥＰが新しいＳＳとして機能するようなバックアップ体制を備えることにより，信頼性の問題を解決している。一方，各々のＥＰ内で実行されるプロセスの管理に関しては，それぞれ独立に行う分散管理方式（このような各ＥＰの制御プログラムをローカル・オペレーティング・システム（ＬＯＳ）という。）を採用した。このような集中分散管理方式を行っているＫＯＣＯＳの制御環境の論理的な関係は，図-6.のような高さの低い円錐形で表わすことができる。円錐の頂点にはＳＳが位置し，底面の円周上には，各ＬＯＳが分散して存在している。各ＬＯＳの管理下にあるプロセス間のコミュニケーションは，ＳＳを介さず，自由に行うことができ，ＳＳとも自由に行うことができることを示めしている。ＫＯＣＯＳの各ＬＯＳはいくつかのユーザプロセス，システムプロセス，および入出力デバイスなどのローカルリソースと共に１つのローカル制御環境を形成している。又、ＳＳおよびＬＯＳで

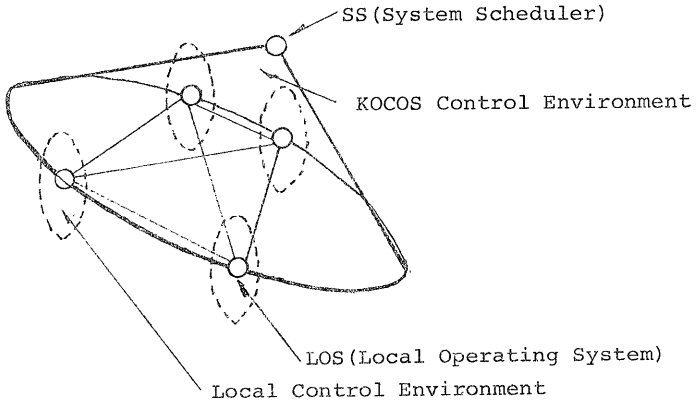
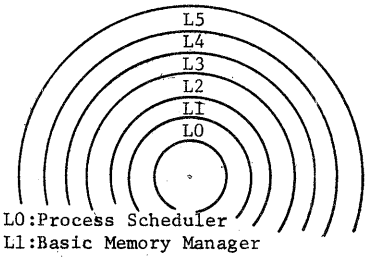


図-6. KOCOS 制御環境の論理的関係図

形成されている円錐は，ＫＯＣＯＳの制御環境を表わしている。

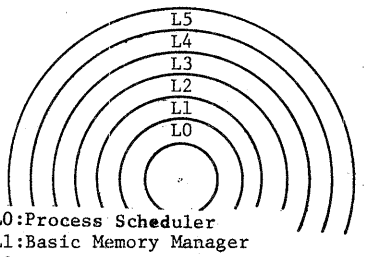
2.2.2. システムソフトウェアの構成

ＫＯＣＯＳでは既存のソフトウェアの有効的活用を考慮しながら，さらにプロセスレベルでの並行処理を考慮したコンプレックス・オリエンテッドなシステムの開発を目指している。又、システムソフトウェアの構成に関しては，高度なモジュール性を持たせ，システムの可変性，拡張性に対処している。ＳＳやＬＯＳの開発および製作において，構造的アプローチ[8][9]を実践し，異種ＥＰとの互換性を考慮して，システム作成用言語[10][11]による設計をし，ポータビリティ，トランスビリティを高めると同時に　信頼性の向上をめざしている。次に，図-7によりＳＳの階層構造を示めし，図-8によりＬＯＳの階層構造を示めす。ＬＯＳの各レベルの機能は，ＳＳの各レベルとほとんど同一の機能なので，ここではＳＳについて概説する。さらに，図-9にプロセス・ステート・ダイアグラムを示めす。



L0:Process Scheduler
L1:Basic Memory Manager
L2:Interprocess Communication Handler
L3:Fault-Tolerant Manager
L4:Basic Monitor Routines
L5:System Command Handler
& System Process

図-7. ＳＳの階層構造



L0:Process Scheduler
L1:Basic Memory Manager
L2:Interprocess Communication Handler
L3:Local Fault-Tolerant Manager
L4:Local Basic Monitor Routines
L5:User Process &
Lower Level System Processes

図-8. ＬＯＳの階層構造

・ＳＳの各レベルの概略機能

レベル０：プロセススケジューラ

リソース管理，ジョブ管理などを行うＳＳ内のシステムプロセスのスケジューリングを行う。スケジューリング・アルゴリズムは，各プロセスのもつプライオリティをもとにしたマルチ・プライオリティ方式である。障害対策用のプロセスには最も高いプライオリティが与えられてる。

レベル１：基本メモリ管理

ＳＳが使用するワーク・スペースのアロケーションや，フリーになったストレッジ・スペースの管理を行う。

レベル２：ＩＰＣハンドラ

ＳＳ内のプロセス間通信の実際の管理を行っている。ローカルプロセスとリモートプロセス間でのメッセージ転送だけでなく，プロセスのコオペレーションに関係するプリミティブをも管理している。

レベル３：障害対策ルーチン

システム内のＥＰやＢＩＵに生じた障害の早期発見，識別，システム再構成，再開始を行う。

レベル４：基本モニタ・ルーチンズ

このレベルでは，ＳＳ下のI/Oデバイス・ハンドリング・ルーチン，リソースやジョブを管理するルーチンなどＳＳの主要な機能が行われる。

レベル５：システム・コマンド・ハンドラ

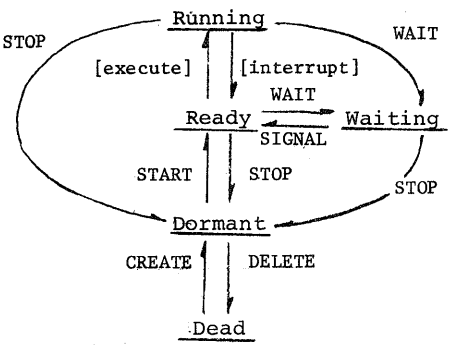


図-9 プロセス・ステート・ダイアグラム

システムコマンドハンドラは，複数のプロセスから形成されていて，ユーザのキーインしたシステムコマンドの受信，解読などを行う。システムコマンドのエラーや要求された動作の返答は，該当ユーザのコンソールへ打ち出される。

2.2.5. システムコマンド

ユーザは，各LOSの管理下にあるローカルコンソールからシステムコマンドをキーインし，KOCOSのサービスを受けることができる。システムコマンドは，LOSによりシンタックスチェックや転送語数のカウントなどの前処理の後，SSへ転送される。SSのシステムコマンドハンドラはそれを解読し，要求された手続きを行い，その結果を該当ローカルコンソールへ知らせる。次に，基本的なシステムコマンドを表-1.に示す。

ユーザがキーインしたシステムコマンドが，どのようなIPC Primitiveを利用してSSのシステムコマンドハンドラへ転送されていくか，簡単なモデルを用いてその処理フローを図-10.の番号にそって解説する。

1. ローカルコマンドハンドラ（＊LCH；各LOSに存在する）が，キーボードリーダ（＊KBR）をwake-upし，FREADを実行する。
2. ＊K.B.Rは，システムコマンドをKey Boardから実際に読み込むための，前処理を行い，1ライン入力完了まで自分自身をブロックする。
3. キーボードからコマンドがキーインされ，＊KBRのバッファにバッファリ

表-1. 基本的システムコマンドの一覧表

コマンド名	オプション	パラメータ	機能
$CON	A	SMN	ローカルな資源をKOCOSのシステム資源として登録する
$DCON	A	SMN	KOCOSの管理下にあるシステム資源をローカルにする
$JOB		JOB-id,pro-n,L'	ユーザ・ジョブを宣言する
$FIN			ジョブ・ステップの終了を宣言する
$END			ジョブの終了を宣言する.
$RES		SMN{:LFN}	ユーザ・ジョブで必要とするシステム・リソースを確保する
$REL	A	SMN{:LFN}	確保したリソースを解放する
$LOAD	L,H	SFN>SMN	システムに登録されているファイルを指定のシステム・モジュールへロードする。
$BRK			ユーザ・プロセスを停止し，システム・コマンド・インプットモードにする。
$RUN			ロードされているプロセスの実行を指示する
$DISP	A,U,R	MN	システム・リソースの状態をユーザへ知らせる.
$MSG	A	Message>SMN	ローカル・コンソールへメッセージを転送する.

コマンドの一般形

$<コマンド名>,<オプション>␣<パラメータ1>,……,<パラメータn>

A:All　L:Local　H:Here　U:User　R:Resource

SMN:System Module Name　IFN:Logical File Name　L':Limit Parameters

SFN:Standard File Name　MN:Module Name

```
"Local Command Handler"
repeat
  SIGNAL(*LCH,*KBR);
1: FREAD(*LCH,*KBR,aa,n,cl);
  syntax check and set parameter;
5: FWRITE(*LCH,*SCH,aa,n,cl);
7: READ(*LCH,*SCH,bb,m,cl);
  print out to local cosole;
forever
```

```
"Key Board Reader"
repeat
2: WAIT(*KBR,*KBR);
  key board release and
  interrupt enable;
  clear(k);
  P(ucb.sem);
4: RANSWER(*KBR,*,buf,n,cl);
  set process name;
  WAITFIM(*KBR,*);
forever
```

```
var buf: buffer area for *KBR;
    k  : buffer pointer
    c  : clock limit

"interrupt from key board"
{get 1 character from Areg}
  buf(k):= Areg;
  k:= succ(k);
  if buf = CR
    then release and RTI;
3: V(ucb.sem);
  RTI;{return from interrupt}
```

```
"System Command Handler
  Receiving process *SCH"
repeat
  WANSWER(*SCH,*,dd,n,cl);
6: semantics check and
    set parameter
  WRITE(*SCH,*,dd,m,cl);
forever
```

図10. システムコマンド内部処理フロー

グされる。読み込んだ文字がＣＲ（キャリッジリターン）であれば，ucbのセマフォ semに対しＶオペレーションを行い，＊ＫＢＲをwake upする。

4. wake-upされた＊ＫＢＲは，＊ＬＣＨへ向けて，RANSWERを実行し，転送完了までブロックする。

5. ＊ＬＣＨは，ＳＳ内のシステムコマンドハンドラの受信プロセス（＊ＳＣＨ）に向けて，FWRITE を実行する。

6. ＊ＳＣＨは dd 番地以降にシステムコマンドを受信し，そのコマンドで要求された処理を行い，その結果を＊ＬＣＨへ向けてWRITEする。

7. ＊ＬＣＨは＊ＳＣＨの返信を受けとり，ローカルコンソールへプリントアウトする。

3. 今後の計画

ＫＯＣＯＳは，プロトタイプの開発だけで終了するのではなく，長期的なプロジェクトである。プロトタイプＫＯＣＯＳを基礎にして，将来的にはユーザの便宜をはかるために以下に列挙するものを開発し，ＫＯＣＯＳを総合システムに発展させる予定である。

[1] 異種間の共有ファイルシステムの開発
ＫＯＣＯＳの形態に適合した共有ファイルシステムの開発

[2] 並列処理
共通言語の開発を含めて並列処理の環境を整備する。

[3] リングサブシステム
種々の入出力機器をリング状に結合し，一括してＣ－Bus に結合する。

[4] デバイス指向のＢＩＵ
プロセッサ用のＢＩＵだけでなく，種々の入出力機器を直接Ｃ－Bus に結合するＢＩＵを開発する。

[5] 大型計算機とのリンク
本学情報科学研究所に設置されているUNIVAC－1106 とＫＯＣＯＳを通信回線を介して結合し，ユーザにより強力なサービスを提供する。

4. おわりに

ＳＳ，ＬＯＳの製作に際し，システム作成用言語を用いて，ソフトウェアの生産性，信頼性の向上をはかっている。しかし，それぞれのミニコンピュータでの割り込み機能の違いなどの物理的環境の細部における相異は，完全に吸収することができず今後の課題といえよう。又，システム全体の信頼性を高めるためにも，各ＥＰにはメモリプロテクション機能，インターナルクロックを備える必要があるといえる。しかし，ＫＯＣＯＳのアプローチにおける特徴は，既存のミニコンピュータに変更を加えずに論理的な結合度を密に複合化できる。又，ＢＩＵの中枢部にマイクロプロセッサを採用し，プロセス間通信の際のオーバヘッドを軽減しただけでなく，高度なメッセージ転送とプロセスの制御が可能な強力なＩＰＣＦにあるといえる。

現在，ＢＩＵ，ＤＢＣは，本年９月末の完成予定であり，ＬＯＳ，ＳＳに関しては，基本設計が終了し，製作中である。

謝辞

ＫＯＣＯＳの開発設計にあたり御協力，御指導いただいた沖電気工業（株）杉浦宣紀氏，松下温氏，さらに本学　相磯秀夫教授，同　土居範久講師，清水洋志氏（現日本航空(株)）に深謝します。又，論文の作成にあたり御協力いただいた本学　阿多靖広君，石井和喜君，瀧塚博志君，西尾忠幸君，福島知善君，松尾繁樹君に深謝します。

参考文献

[1] 元岡達：コンピュータコンプレックスの展望，情報処理，vol.15，no.7，pp.525～533，1974.

[2] Lochkeed Electronics Company : SUE Computer Handbook, Los Angeles, 1972.

[3] 上林，徳田，竹山，石塚，西垣，平塚：KOCOSのアーキテクチャ(1)－フィロソフィとシステム構成，情報処理アーキテクチャ研究会資料，1975年7月.

[4] Dijkstra, E. W. : Cooperating Sequential Processes, Programming Languages, Academic Press, pp.43-111, 1968.

[5] Walden, D. C. : A System for Interprocess Communication in a Resource Sharing Computer Network, Comm. ACM,vol.15,no.4, pp.221-230, Apr. 1972.

[6] Akkoyunlu, E., Bernstein, A., Schantz, R. : Interprocess Communication Facilities for Network Operating Systems, Computer,vol.7,no.6, pp.46-55, Jun. 1974.

[7] Akkoyunlu, E., Bernstein, A., Schantz, r. : An Operating System for Network Environment, Proc. Synposium on Computer-Communications Networks and Teletraffic, Polytechnic Press, New York, pp.529-538, 1972.

[8] Dijkstra, E. W. : The Structure of the "THE" Multiprogramming System, Comm. ACM,vol.11,no.5, pp.341-346, May 1968.

[9] Hansen, P. B. : The Nucleus of a Multiprogramming System, Comm. ACM,vol.13,no.4, pp.238-241, Apr. 1970.

[10] Wirth, N. : The Programming Language PASCAL, Acta Informatica, vol.1,no.1, pp.35-63, 1971.

[11] Wulf, W. A., et at., : BLISS Reference Manual, Computer Science Department Report, Carnegie-Mellon Univ., Pittsburgh, 1971.