EC77-43

網向きオペレーティングシステムについての一考察

Network Oriented Operating System

和賀井 フミ子　田中 英彦　元岡 達
Fumiko WAGAI　Hidehiko TANAKA　Tohru MOTO-OKA
東京大学 工学部
University of Tokyo

序章 はじめに

近年コンピュータ・ネットワークの研究が盛んに行なわれるようになったが、我々の研究室においても、数年前から、実験的に計算機網を構成し、網について総合的な研究を行なっている。その一環として、計算機網に向いたOSはいかに構成したらよいか、を研究し、実装及び実験、評価を行なっている。

ここでは研究用計算機網TECNET上に実装された計算機網向きオペレーティングシステムNOSについて、基本思想、実装及びその問題点と改良について述べ、計算機網向きOSのあり方を考察する。

第1章 NOSについて

第1節 NOSの目的

TECNETは、使用目的(及びメーカ)の異なる3台の計算機を、汎用性を考慮して通信回線によって接続した網である。このような計算機網の下で有効に機能することを目的としてNOSが作成された。特に考慮した点は、計算機が複数台在ることにいかに対処するか、であった。従来の単一計算機に対して作られたOSは、一台の制御のみ行なえばよい、が網向きOSでは、単に一台の制御のみでなく、複数台の計算機にまたがった制御も必要となる。しかも、使用者には使いやすい形にしなければならない。網内の計算機毎に異なる方法で資源にアクセスするのでは、不便であり、網を構成した意味が薄れる。NOSでは、この点を重くみて、網内のどの計算機に対しても同じ形式でかつ能率良く資源を使用できるように、OSを作成した。このことは、計算機の境界が取り除かれたことと同等の意味を持つ。計算機の置かれた場所や、形の違いを意識することなく、自在に計算機を利用できるので、一台の計算機だけでは実現できなかった様々な新しい使用法が、使用者の負担を増すことなく、可能となる。このような一様制御ができる為には、プロセス間通信方式が網上で統一されていることが必要である。NOSではこのような統一方式を採用している。

以上述べたように、NOSは、計算機網に適したOSすなわち、網の利点を生かすOSのあり方を研究する為に作られたOSであり、上記のようなプロセス間通信機構を下部に持つ。その上部には、網向きファイル制御や網向きジョブ制御など網システム利用に必要な制御機能を実現するプロセス群が構成されている。

NOSは、以上のようなシステム構成の手法やシステム構造を検討することを目的としている。

第2節 TECNETの構成

TECNETは、図1-1及び表1のように構成されている。

第3節 NOSの構成

OSは、仕事を行なうものとしてのプロセスの概念によって構成されている。プロセスは計算機制御の軌跡の単位の一つで、スケジューラの対象となるものである。プロセス実行の場とし

て核がある。プロセスはプリミティブを(道具として)用いて、核の機能を利用し、種々の動作を行なう。すなわち、NOSでは、OSがプロセスと、プロセス実行の場を提供するものとしての核との2階層から構成され、プロセスと核とのやりとりは、プリミティブを用いて行なわれる。

核には次のような機能がある。

1　プロセス間通信機能
2　マルチプログラミング機構
3　プロセスの生成・消滅
4　割込処理

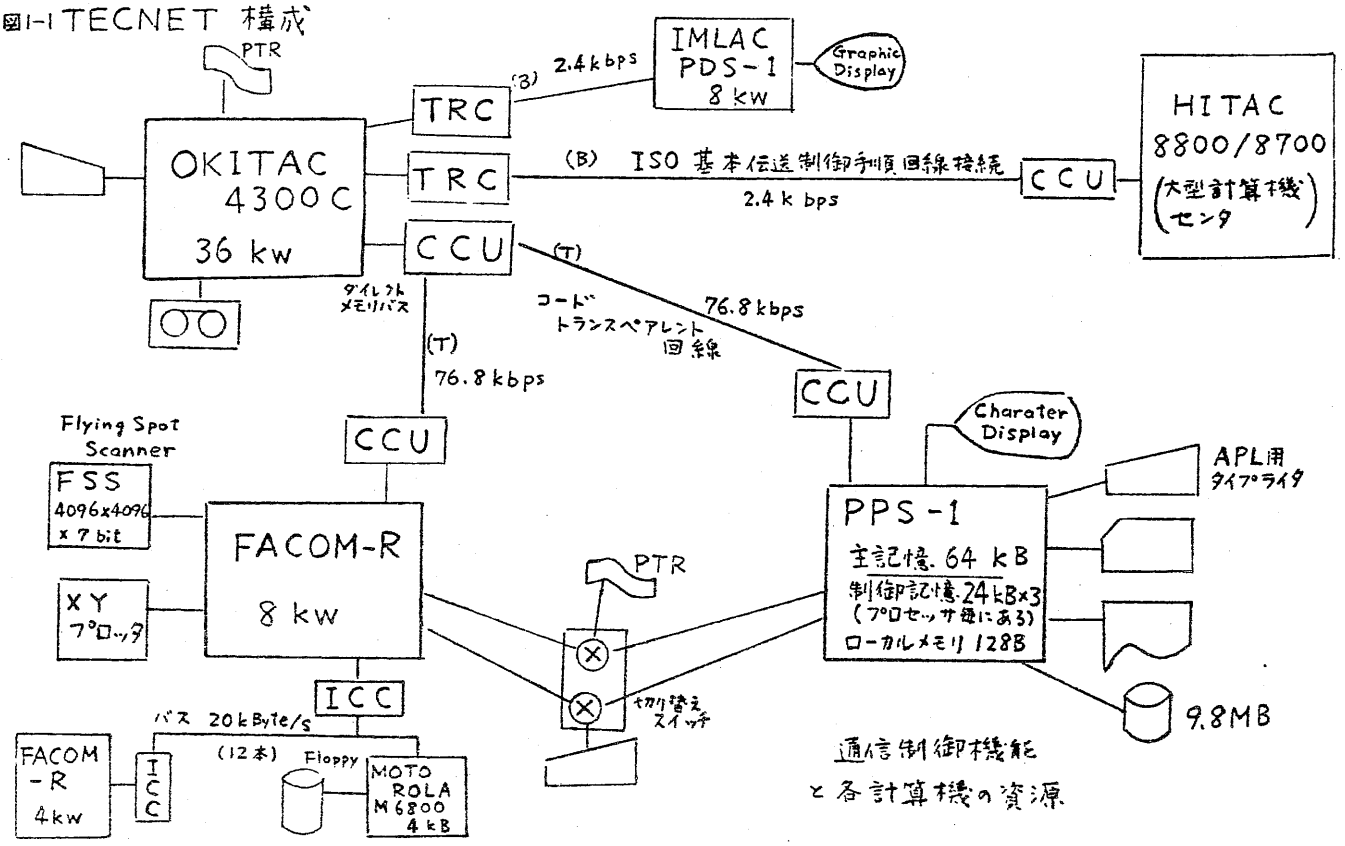
1は、本OSでは重要な役割を果たす機能である。すなわち、プロセス間の協調方式は、すべて、プロセス間通信プリミティブを用いたプロセス間通信方式によっている。この方式では計算機網に適するように、ローカル通信、リモート通信にかかわらず、一様の制御が行なえる、という特徴がある。つまり、プロセスは、相手プロセスがどの計算機に存在していても、同じ形式で通信プリミティブを用いることができる。また この方式では、両プロセスが通信の約束を取り交わし、合意が成立してから、実際のデータの転送が行なわれるようになっている。

この部分の詳細は第2章以下で述べる。

2は単なるI/O待ちによるプロセス・スイッチングではなく、各Readyプロセスに対して、タイムスライス制御によるCPUの割当てを行なっている。すなわち、プロセスは、入出力待ちに関係なく並列に処理される。外から見ると幾つかの仕事が、同時に進行する。このことをCPU一台で可能にする為に、プロセスの処理を時間で区切る方法を取っている。

NOSには、時間の概念があり、時計を持っている。すなわち、ハード的にTIMERを持ち、これが一定時間毎に割り込んで、時計の時刻を更新している。各プロセスには、タイムスライスが

図1-1 TECNET 構成



計算機名	機種	メモリ サイズ	NOS	使用目的・資源
PPS-1	ポリプロセッサ・システム (主記憶を共有したプロセッサが3台有り、同時に異なる処理を行なう)	主記憶 64 kB 制御記憶 24kB ×3 ローカルメモリ	NOSの主処理	諸言語プロセッサ PLAN PASCAL APL
OKITAC 4300C	ミニ コンピュータ	36 kw (1w=2B)	NOSの処理	大型計算機センタへのインタフェス IMLAC 及び CMTと接続
FACOM-R	ミニ コンピュータ	8 kw (1w=2B)	OKITACの縮小版	各種の端末制御 FSS・TTY・XYプロッタ・フロッピディスク

表 1

与えられ、プロセス自身は知らないが、核ではTIMER割込時に、プロセスの一回の処理時間がタイムスライス内になるように監視している。

3について述べる前に、プロセスの状態のことに触れる必要がある。

プロセスには、Run, Ready, Wait, Dormant, Dead の5つの状態がある。プロセスは主にプリミティブを用いて状態遷移を行なう。

プロセスにCPUが割り当てられている状態をRun状態という。プロセスがRun状態からWait状態になるのは、自身がBLOCKプリミティブを発行して、仕事を止める場合である。あるプロセスがWait状態になると、核の一部であるディスパッチャに制御が移る。ディスパッチャではReady状態にあるプロセス群の中から、最も優先度(仕事,時間など)の高いプロセスを選び、これをRun状態にして、処理を開始させる。前記2のようにタイムスライスを使い切って強制的に中断させられた場合にも、ディスパッチャに制御が移る。中断プロセスはReady状態になる。Ready, Wait, または Dormant状態にあるプロセスに、他からSTARTプリミティブを用いて起動をかけると、Ready状態になる。起動をかける方法は、他のプロセスや割込処理が必要に応じて任意にかける場合と、プロセス自身がTIMEプリミティブを発行し起動時刻を指定して、指定時刻にTIMER割込から起動をかけてもらうやり方とがある。あるプロセスが他のプロセスにSTOPプリミティブを用いると、Dormant状態になる。

プロセスの生成・消滅は、このようなプロセスの状態遷移の一部分である。あるプロセスが、CREATEプリミティブを発行すると、プロセスが新たに生成する。生成されたプロセスは、Dormant状態になる。逆にREMOVEプリミティブを用いると、プロセスを消滅させることができる。

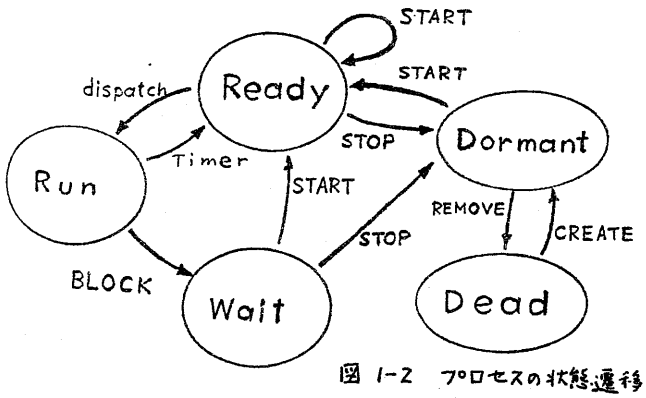


図 I-2 プロセスの状態遷移

4は、割込の処理方法とそれをプロセスにどう翻訳するかについて、である。

割込には、入出力端末からの割込, 通信回線からの割込, TIMER割込, CPUハードウェア異常メモリパリティ・エラなどによる割込などがある。

割込は重要度に応じて幾つかのレベルに分けられている。割込処理では、各レベルに応じた処理を行なっている。入出力端末からの終了割込や、回線からの割込では、処理プログラムは割込の内容を解析し、端末対応プロセスに情報を与える必要があれば、このプロセスに起動をかける。割込処理プログラムでは、割込の内容を解釈し、それを入出力関係のプロセスに伝える役割を荷っている。従ってプロセスは、実際に起こった割込については全く関知しない。

TIMERによる割込については、先にも少し触れたが、この処理では、時計の時刻の更新と、Runプロセスのタイムスライス値のチェックとその処理, 及びTIMEプリミティブを用いて起動時刻を指定したプロセスの起動の可否を調べ、可ならこのプロセスに起動をかけることを行なっている。この場合もプロセス自身は、割込が起こっていることを全く知らないままである。

CPUのハードウェアの異常や、メモリパリティエラ等が生じた時は処理プログラムはオペレータ・コールを出す。

以上述べたように、プロセスは、割込とは全く無関係であり、割込はすべて、核において解釈・処理される。

プロセスは、機能的には 核プロセス, システムプロセス, ユーザプロセスから成る。

核プロセスは、核の機能を援助するプロセスであり、通信制御プロセス, Pager等がある。システムプロセスは、ロガー, 網向きファイルプロセス, 網ジョブ管理プロセス等から成る。網ファイルプロセスは、入出力端末を一様にファイルと見なしてアクセスできるように作られたプロセスで、このプロセスには、各端末に対するFDPと、これらを制御するFCPとがある。網ジョブ管理プロセスは、計算機網向きに作られたジョブ管理を行なうもので、計算機の境界を意識せずに使用することができる。これにはMJMPと幾つかのLJMPなどがある。このプロセスの詳細については、参考文献Ⅳに述べられ

ている。

核プロセスは、その機能上、他のプロセスとは異なって、通信プリミティブは使えないプロセスである。（それ以外のプリミティブは使用できる。）

核とプロセスとのインタフェスに用いられるプリミティブについて、その機能と使い方を少し述べる。プリミティブはプロセスからは自由に利用できる基本的なオペレーションであり、一種のシステムコマンドのように見える。また、ユーザプログラムからは、機械語の拡張と見ることのできるものである。プリミティブは割込禁止の状態で動作する。プロセスがプリミティブを使用する時は、必要な引数を設定して、サブルーチン・コールの形で用いる。

プリミティブは、機能的には次の4つに分けられる。

1　プロセス間通信用プリミティブ

プロセス間の協調を行ないたい時には、SENDとRECEIVEプリミティブを用いる。

2　プロセスの動作制御用のプリミティブ

プロセスの状態遷移を行なう時に用いる。STARTプリミティブは、指定されたプロセスを、Ready状態に変えるプリミティブ，BLOCKは、Run状態のプロセスが用いて、それ自身をWait状態にするプリミティブ，STOPは、他のプロセスをDormant状態にするプリミティブ，TIMEは、プロセスの起動予定時刻を核に登録するプリミティブである。

3　プロセスの生成・消滅用のプリミティブ

プロセスは、CREATEとREMOVEプリミティブによって、生成・消滅される。

4　情報センスプリミティブ

プロセスが、核からの情報を得るには、SENSEプリミティブを用いる。SENSEを発行すると、主に自分のプロセス通信に関した情報が得られる。SENSEは1と組合せて用いると、威力を発揮する。

プリミティブの中には、それ自身で他のプリミティブを使用できるものもある。また、プリミティブを組み合わせたマクロの形も使われる。プリミティブ一つでは、基本的な機能しか持っていないが、これを組み合わせることによってより複雑な制御を、使用者の負担を増すことなく、行なうことができるようになる。

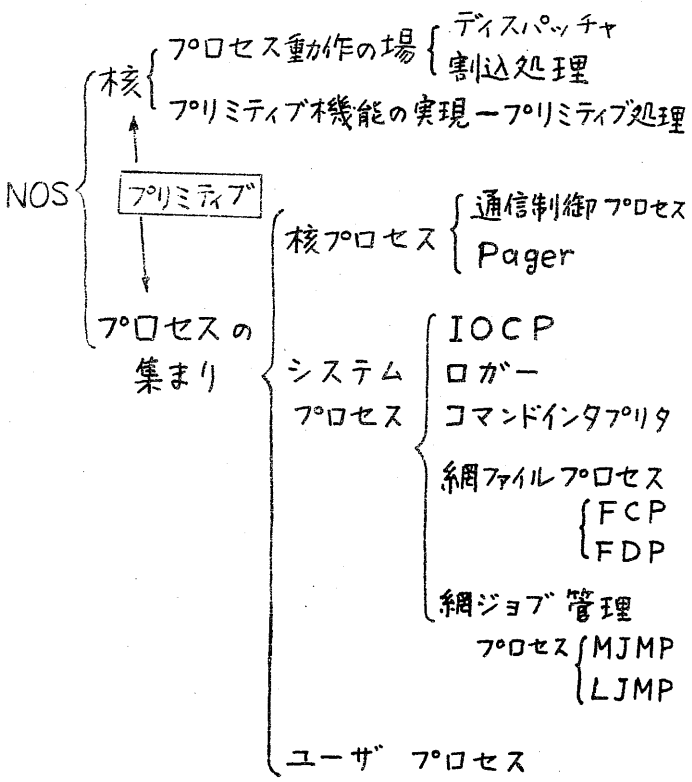


図 1-3　NOSの構成

第4節　NOSのバーチャルマシン

NOSにおいては、次のような多階層より成る仮想マシンを構成していると考えられる。

レベル

0　マイクロ・コード
1　機械語
2　核
3　核プロセス，START, BLOCKなどのプリミティブ
4　通信プリミティブ
5　システムプロセス
6　ユーザ プロセス

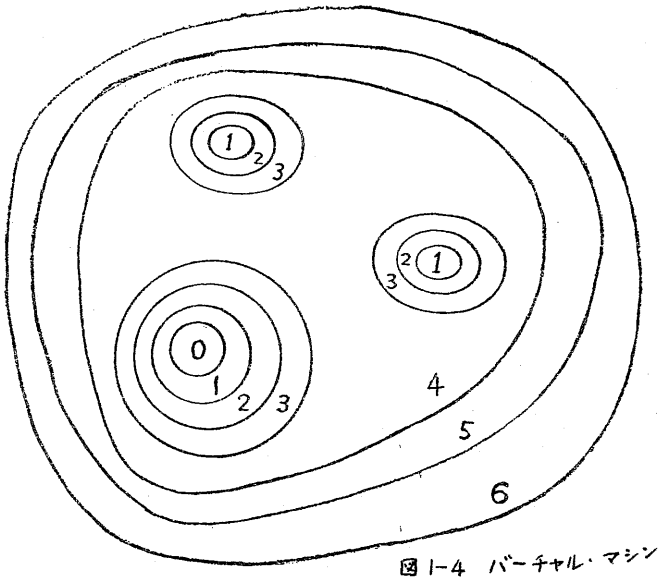


図 1-4　バーチャル・マシン

第2章　プロセス間通信システム

NOSのプロセス間通信は、異なる計算機に対しても統一した形で通信を行なえるシステムである。プロセスは、相手がどの計算機に存在するプロセスであっても、同じ手法で通信を行なうことができる。NOSに実装されているプロセス間通信の構造は、使用者レベルから見た場合のインタフェースと、その下位にある核間プロトコル，更に下位レベルの回線上の伝送手順；の3階層から成っている。この章では、これらの各階層について、詳しく述べ、システムの構造を明らかにする。　尚、ここでのプロセスの状態は、通信プリミティブ提出の時点では、Run状態になっている。

第1節　ユーザ・インタフェス

プロセス間で情報を伝達するには，プロセスは，それぞれに，通信プリミティブSENDと，RECEIVEを提出する必要がある。データ送信側のプロセスは，SENDを用い、受信側のプロセスはRECEIVE を用いる。提出形式は図2-1 に示される。各計算機に存在するプロセスは，相手プロセスの存在する計算機が 同じであっても，異なっていても、通信プリミティブは同じ形式を用いる。つまりプロセスは計算機の境界を意識する必要はない。SENDとRECEIVEはどちらを先に提出してもよい。通信相手が不定である受信側のプロセスは，RECEIVE ANY を用いる。両プロセスがSENDとRECEIVEを出し合うと、通信契約が成立し、データの転送が行なわれる。或るプロセスがSENDを出しても、相手プロセスがRECEIVEを出さなければ、いつまで経ってもデータの転送は行なわれない。

プロセスはデータの転送 が終了するまで Wait状態で待つことができる。通信プリミティブの後でBLOCKを発行すればよい。この場合は、転送が終了すると，プロセスに起動がかかる。プロセスが、データの転送が終了したことを確認するには，SENSEプリミティブ を用いる。もしも受信側のプロセスが SENSE を用いた時に、転送一部終了・更に×Byte のバッファが必要という情報が得られたら、受信バッファ を用意して再びRECEIVEを提出する必要がある。というのは，この時には送信データ長よりも受信バッファ量の

図2-1　通信プリミティブ提出形式

♦ OKITAC内に存在するプロセスの提出形式

CAL	SEND	CAL	RECV
DC	相手Host番号	DC	status・相手Host番号
DC	送・受プロセス番号	DC	送・受プロセス番号
DC	送信データアドレス	DC	受信バッファアドレス
DC	送信データ長(byte)	DC	リスタートロケーション
J	エラー時の行き先	DC	受信バッファ容量
		J	エラー時の行き先

RECV = RECEIVE

RECVでは statusによってRECEIVE & BLOCKができる

♦ FACOM-Rに存在するプロセスの提出形式

BL	R SEND	BL	R RECV
DC	相手Host番号	DC	相手Host番号
DC	送・受プロセス番号	DC	送・受プロセス番号
DC	送信データアドレス	DC	受信バッファ先頭アドレス
DC	送信データ長(byte)	DC	リンク番号

♦ PPS-1のプロセスについては、参考文献II 参照

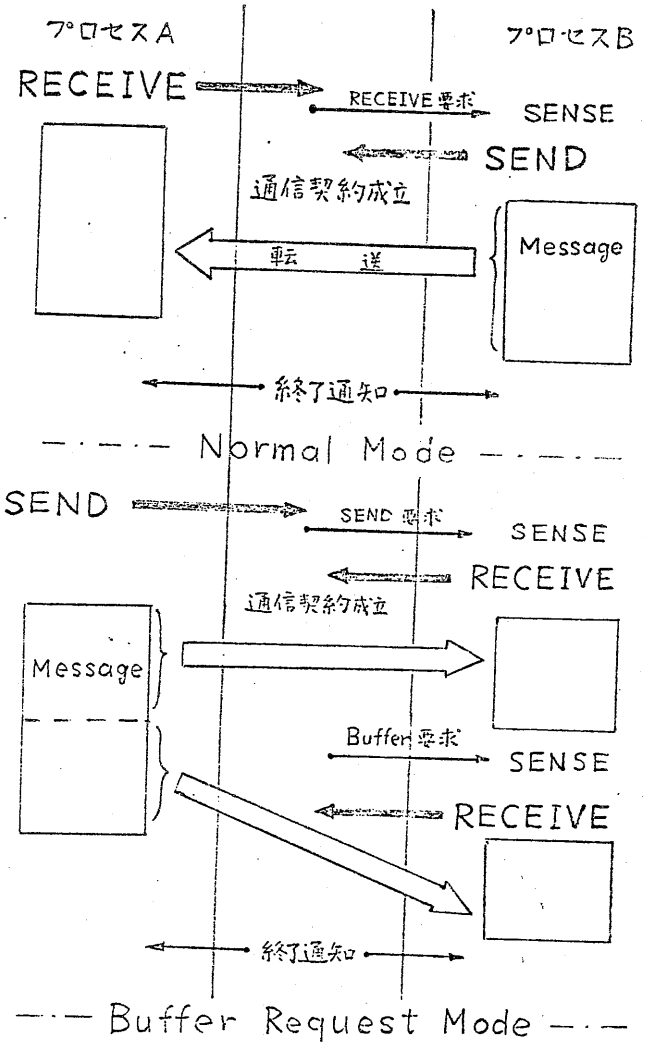


図2-2　ユーザインタフェスによるプロセス間通信の概念

方が小さく、転送は、受信バッファの分だけしか行なわれていないからである。(バッファ要求モード) バッファ要求モードのRECEIVEが提出されると、直ちに新たに用意できた受信バッファの分だけ、データが転送される。バッファ要求モードのRECEIVEは、データの転送が完全に終わるまで、何度も使用できる。送信側のプロセスは、バッファ要求モードには、何ら関知しない。

第2節　核間プロトコル

ここでは主に、核、と核プロセスである通信制御プロセスの働きについて述べる。ローカルとリモートの差違は、この段階で吸収される。

I　プロセス通信に用いるテーブル (主なもの)

CSW　プロセス通信に必要な情報を書き込んでおくテーブル

IW　主にプロセス通信の進行状況を書き込むテーブル。SENSEでは、ここから情報を読み出す。

II　ローカル通信の構造

i) 通信プリミティブ SEND/RECEIVEの動作

先に提出したプリミティブは、CSWとIWを確保して、CSWにプリミティブの引数を書き込み、相手側のIWに通信要求のある旨を書いて待つ。後から提出したプリミティブでは、通信要求を出したCSWを探し、通信契約を取り交わし、CSWにプリミティブの引数を書き込んで通信制御プロセスに起動をかける。

ii) 通信制御プロセスの処理

通信制御プロセスでは、契約成立でローカル用のCSWを探し、ここに書き込まれている送信アドレスから受信バッファへ、送信長と受信量のうち大きくない方の長さだけ、データを移し変える。これで完全に転送が終わりなら、送信用と受信用のIWに終了通知を書き、CSWを空にしてリンクを切る。転送すべきデータがまだ残っている場合は、受信側のIWにその旨書く。

iii) バッファ要求モードの場合

受信プロセスがSENSEによって、転送一部終了且バッファ要求であることを知り、受信バッファを用意して再びRECEIVEを出すと、RECEIVEプリミティブではCSWの一部を書き直して、通信制御プロセスに起動をかける。この後の処理は、ii)と同じである。

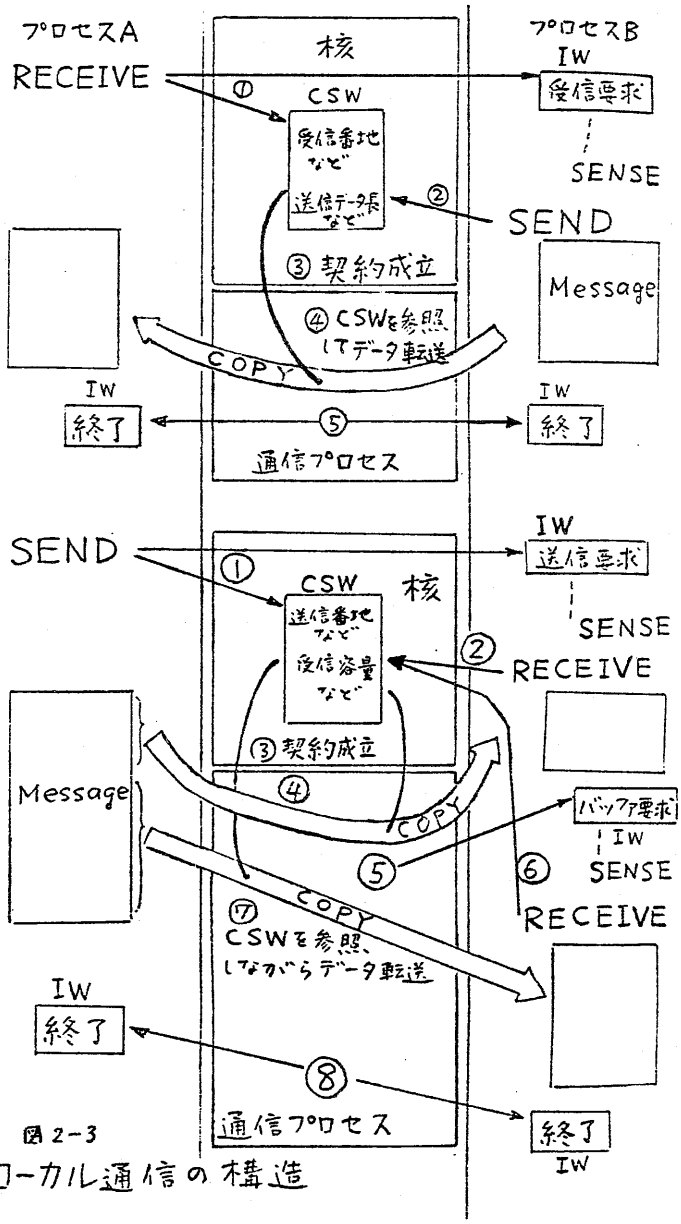


図2-3
ローカル通信の構造

III　リモート通信の構造

i) ローカル通信との差違

リモート通信においても、ローカル通信の場合と同じように、通信制御プロセスは、CSWの値によって、メッセージの転送処理を行なう。従って、送・受プロセスの存在する各々の計算機にCSWを置く必要がある。各CSWには、両計算機にある通信プリミティブからの情報が書き込まれる。通信契約は両計算機内で取り交される。　リモート通信の場合は、通信プリミティブや通信制御プロセスからの情報やメッセージは、伝送路を通して送る操作が必要である。この為に、制御情報伝達にはコマンドを用いて、メッセージはパケットに分割して転送する。

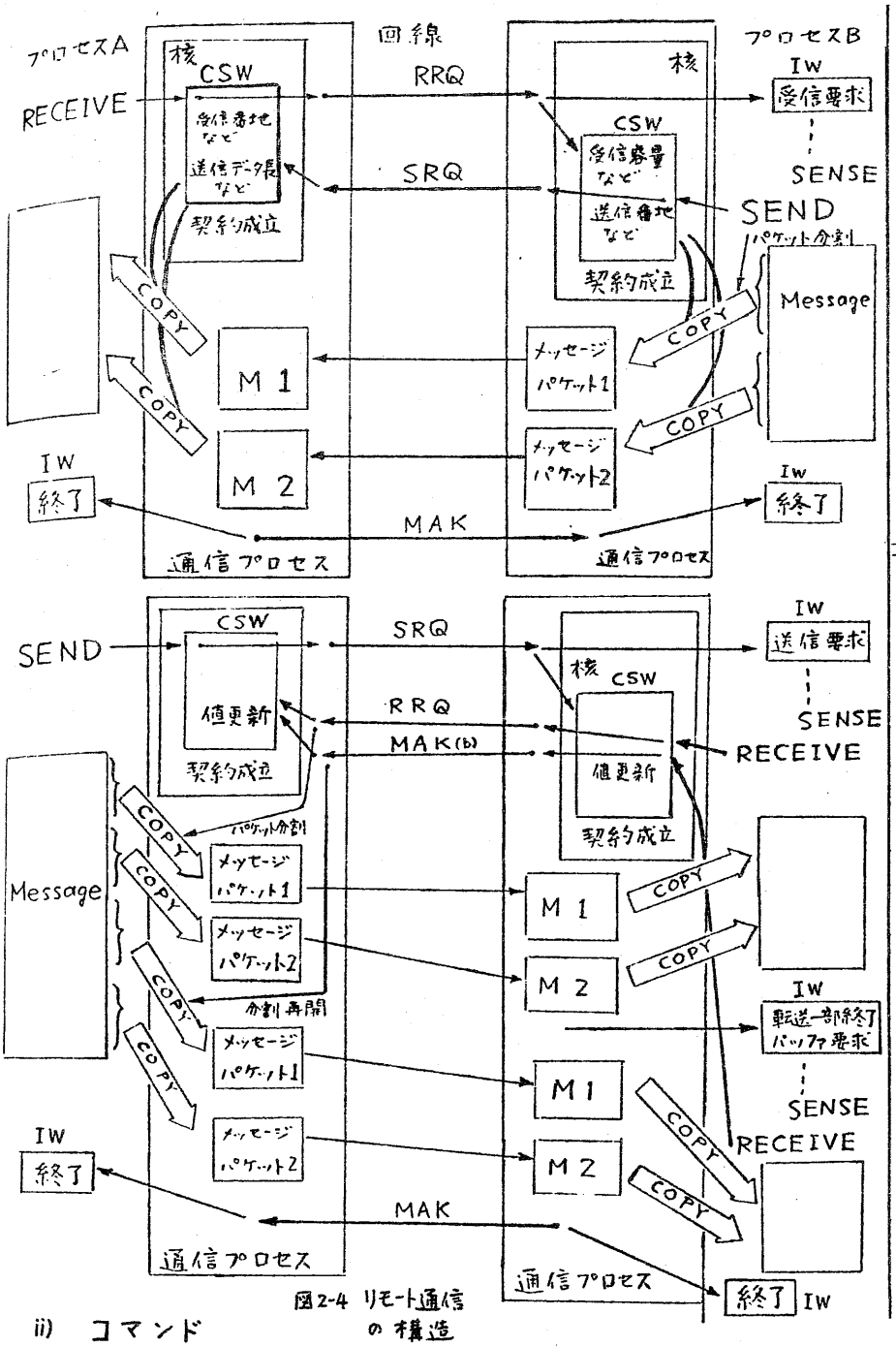


図2-4 リモート通信の構造

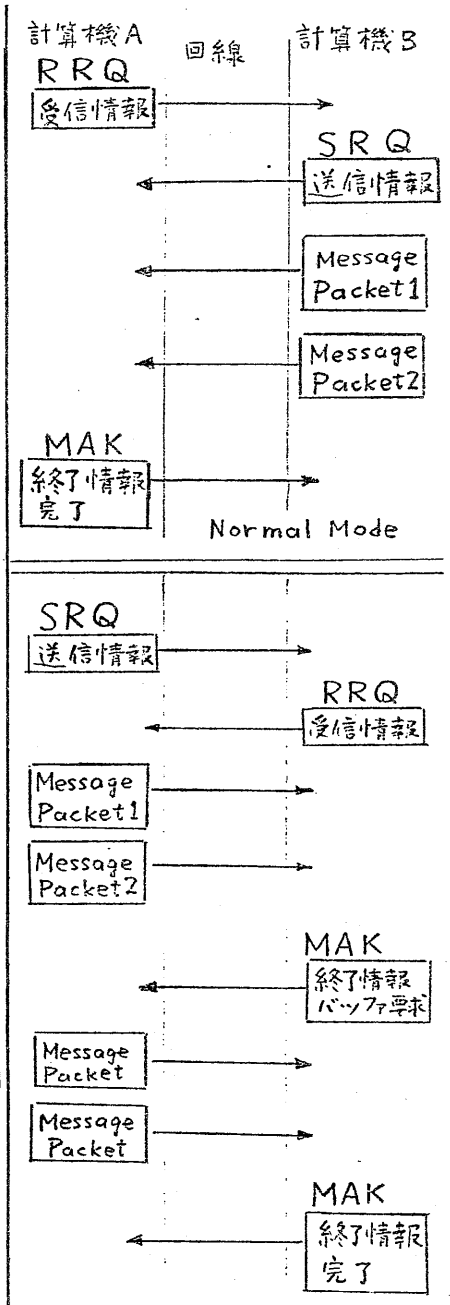


図2-5 通信プロトコル

ii) コマンド

コマンドは、計算機間でプロセス通信に関する情報を伝達するものである。通信プリミティブを補助するものとして、SRQ, RRQ, 通信制御プロセスに対しては、MAK, CLOSがある。

SRQは相手プロセスにSENDを提出したことを知らせるコマンドであり、RRQはRECEIVEを出したことを知らせるものである。MAKはメッセージの受信が終わったことを、送信側のプロセスに知らせるもの。CLOSは、通信制御プロセスの都合で通信が続行できなくなった時に用いるコマンドで送信側と受信側の両通信制御プロセスの合意によって通信を打ち切っている。

iii) メッセージのパケット分割と合成

伝送路上では、メッセージを一定長の長さに分割して送っている。この分割の単位をパケットと呼ぶ。各パケットには数Byteのヘッダがつけられ、行き先、メッセージ別、等が識別できるようになっている。

受信側では、受信パケットをメッセージ毎に分類し、一メッセージに対しては、全てのパケットが揃ってから、受信バッファに書き込む。

iv) プリミティブの動作

リモート通信では、通信要求を伝送路に送り出す操作が増えている。すなわち、先に提出したプリミティブでは、自分用のIWとCSWを確保し、CSWにプリミティブの引数を書き、次にSRQ又はRRQを作る。後から提出したプリミティブは、(SRQ又はRRQによって作られた)

CSWを探し、ここで通信契約を交し、通信制御プロセスにはこの旨を知らせて、SRQ又はRRQを作る。リンク番号はRECEIVEプリミティブが決める。RECEIVE ANYを先に提出した場合は、SRQを受信してから、RRQを送る。

v) 通信制御プロセスの処理

送信側の処理　SENDを先に提出した場合 ① SRQを回線に送り出す。② RRQを受信して、通信要求を出したプロセスのCSWを探し、通信契約を交す。③ 通信契約が成立したら、メッセージをパケットに分割し、それぞれを回線に送り出す。④ MAKを受信する。その情報が完全転送済であれば、IWに終了通知を書きCSWを空にして、リンクを切る。

RRQが先に到着した場合　① IWとCSWを確保し、CSWにRRQからの情報を書き、IWに通信要求を書く。②（SENDが出ると、この段階で通信契約が成立するが、）SENDによるSRQを回線に送り出し、上記③以下の処理を行なう。

受信側の処理　RECEIVEを先に提出した場合 ① RRQを回線に送り出す。② SRQを受信したら、該当のCSWを探して、通信契約を交し、CSWにSRQからの情報を書く。③ メッセージパケットが到着したら、これをメッセージ毎に区分けする。④ 一つのメッセージに対するパケットが全部揃ったら、受信バッファにパケットの内容を一パケットずつ書き込む。⑤ 全パケットの内容の移し変えが済んだらMAKを作り、回線に送り出す。IWに終了通知を書いてCSWを空にし、リンクを切る。

SRQが先に到着した場合　① SRQを受信したらCSWとIWを確保し、CSWにはSRQからの情報を書き、IWには通信要求を書く。② (RECEIVEが出ると、通信契約が成立するが、) この時のRECEIVEによるRRQを回線に送り出し、受信側の③以下の処理を行なう。

vi) バッファ要求の場合

パケット分割して転送するメッセージ長は、送信長と受信量のうち、大きくない方である。v) 受信側④の処理が終わった時に、送信長と受信量を比較し、受信量の方が少ない時は⑤に移らずに、IWにバッファ要求である旨を知らせる。プロセスが受信バッファを用意してRECEIVEを出すと、このプリミティブは、CSWの値を一部変え、MAKを作る。通信制御プロセスは、MAKを回線に送り出す。送信側では、通信制御プロセスがMAKを受信すると、MAKの情報をCSWに書き、再びv) 送信側③以下のことを行なう。送信側のプロセスは、このことに、全く関知しない。

第3節　回線レベルのプロトコル

I　ACKと再送

コマンドやメッセージのパケットは、回線を通して転送される。確実に相手計算機に送られたことを確認する方法として、パケット受信側では、受信パケットのヘッダをコピーし、返事である旨のフラグをたてて送信元に送り返している。（ACKパケット）

パケット送信側では、パケット送出時にTIMEプリミティブを用いて、或る時間後に、再送プロセスの起動を依頼する。指定時刻が過ぎてもACKが来ない場合は、再送プロセスは、このパケットを再び回線に送り出す。（再送処理）

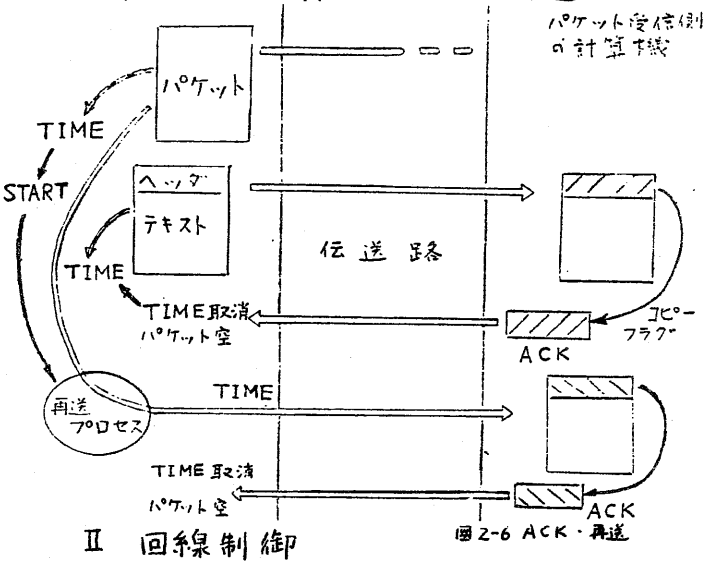


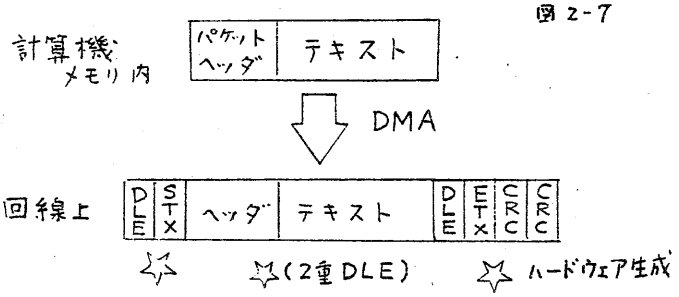
図2-6 ACK・再送

II　回線制御

二重DLE法によってコードトランスペアレントな伝送が可能となっている。

エラーのチェックは巡回符号を用い、ハード的に、符号の付加，及びエラーの検出を行なっている。

図2-7



回線上のパケットは、データパケット，ACKパケットを問わず、全て上の形をしている。

第4節　実装

ここでは、TECNET 構成計算機のうち、OKITAC 4300C と FACOM-R に実装されたプログラムについて、述べる。PPS-1 については、参考文献Ⅱを参照していただきたい。

Ⅰ　プログラムの大きさ

計算機		本体	プリミティブ	プロセス
OKITAC 4300C	a プロセス間通信関係		SEND 0.25 kw (1ページ) RECEIVE * 0.5 kw (2ページ) SENSE 0.25 kw (1ページ)	通信制御プロセス {ローカル 0.25 kw (1ページ) リモート 2 kw} 再送 0.12 kw
	b その他	ディスパッチャ 0.25 kw 割込処理 全部で 2.25 kw {入出力 1.5 kw TIMER 0.25 kw ハード異常 0.5 kw}	START 0.5 kw BLOCK 0.13 kw STOP 0.2 kw TIME 0.25 kw CREATE 0.2 kw REMOVE 0.25 kw	コマンドインタプリタ 1 kw ロガー 0.25 kw IOCP {TW 0.5 kw PTR 0.25 kw IMLAC 0.3 kw など
FACOM-R	a		SEND RECEIVE 合わせて 0.34 kw	通信制御 (リモート処理のみ) 2.3 kw
	b	モニタ 66w		コマンドインタプリタ 0.44 kw TW プロセス 0.37 kw FSS 約 2 kw

＊ OKITAC にはページの概念がある。1ページ=256w

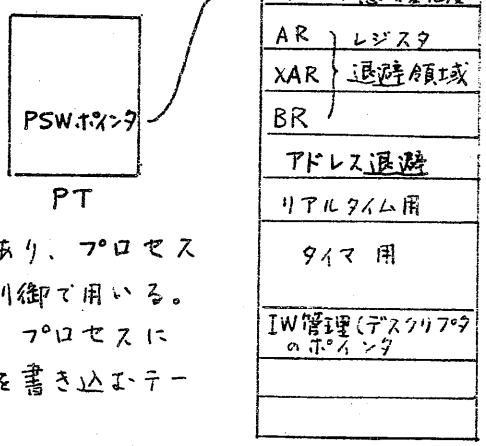
OKITAC 内に実装されている全プロセスのプログラムの大きさを合計すると 12.8 kw になる。

Ⅱ　使用テーブルの構造

i)　PT と PSW

プロセステーブル(PT)は、プロセス名などを書き込んでおくものであり、プロセス状態遷移の制御で用いる。

PSW は、プロセスに関する情報を書き込むテーブルである



ii)　CSW

OKITAC に実装された CSW

IW ↓ ↑

ローカル用	リモート送信用	リモート受信用
status 1	status 1	status 1
status2・相手Host#	status2・相手Host#	status2・相手Host#
送・受プロセス#	送・受プロセス番号	送・受プロセス番号
受信バッファアドレス	リンク番号	受信バッファアドレス
リ・スタートロケーション		リ・スタートロケーション
受信バッファ容量	受信バッファ容量	受信バッファ容量
送信データアドレス	送信データアドレス	リンク番号
送信データ長	送信データ長	送信データ長

status 1

bit 0 =1 使用中, =0 空き
bit 1 =1 ローカル通信, =0 リモート通信
2 =1 RECEIVE用テーブル, =0 SEND用
3 =1 契約成立
4 =1 データ転送中 (最初のACK受信で0になる)
5 リンクモードで一部終了
6 未転送有 (バッファ要求モード)
etc

status 2

bit 6 RECV側でBLOCK指定有
etc

FACOM-R では、リモート通信のみ実装したのでCSWは、右の2つの形だけである。

iii)　IW

SENSE情報
Host#, Process#
データ長

SENSE情報 (bit 0 Null, 1 未SENSE有) SENSEプリミティブで書く

bit 2 エラー
3 事象発生
4 通信終了
} コードは bit 12〜15 に詳しい内容
5 リンクモード
6 バッファ要求

IWはディスクリプタを用いて管理される

iv)　その他のテーブル (リモート通信用)

回線制御用に、送信LCUテーブルと受信LCUテーブルがある。通信制御プロセスは、メッセージのパケット分割・合成の為の管理テーブルを持っている。

Ⅲ　コマンドとメッセージのヘッダ

コマンド

0	1	2	3	4

メッセージ

0	1	2	3	4	メッセージテキスト

コマンド	メッセージ
0 word 自Host#, 相手Host#	0 自Host# 相手Host#
1 送・受プロセス番号	1 パケット番号, メッセージ番号
2 status 3	2 status 3
3 {SRQ: 送信バイト長 RRQ: 受信容量 MAK: バッファ要求のみ 受信バッファ容量}	3 (パケット番号=1なら) リンク番号
4 RRQ: リンク番号	4 メッセージテキスト

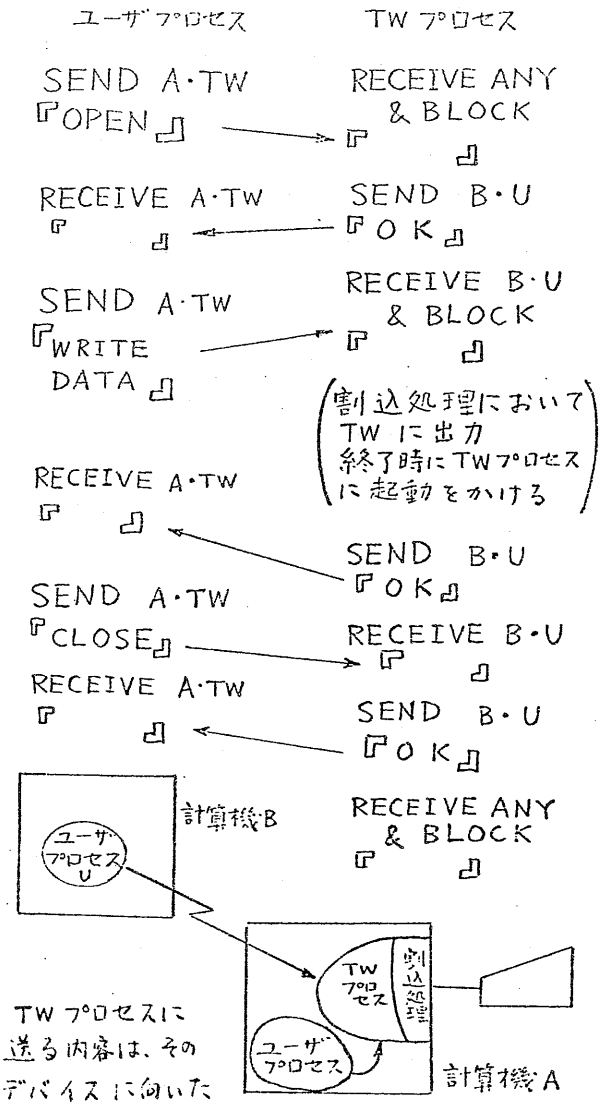
status 3

bit 0 コマンド表示
bit 1 SRQ
2 RRQ
3 CLOS
4 MAK
:
12 バッファ要求
14 ACK

メッセージパケットはヘッダこみで 256w から成る。

第5節　使用例

簡単な例として、或るユーザUが計算機Aにあるタイプライタにデータを出力する方法を述べる。



TWプロセスに送る内容は、そのデバイスに向いた指令であり、一般にデバイス毎に異なる。

第3章　現プロセス間通信の問題点とその改良

第1節　デッドロックの問題とその解決策

以上述べたようなプロセス間通信方式を用いて、様々な使用実験を行なったところ、次のようなデッドロックが生じた。

1　通信するプロセスがお互いの相手に対して、SEND-BLOCK を出し合いRECEIVEが出せないので、ともにWait状態のままになる。

2　プロセスどおしの同期ずれ（つまりBLOCKとSTARTプリミティブの時間のずれ）のためにいつまでもWait状態が解けない。

これらの問題は単一の計算機システムでも問題になることではあるが、計算機網の場合は、プロセスが互いに離れたシステムに存在しているので、特に顕在化する。すなわち、計算機間の情報伝達遅延が原因であって、BLOCKプリミティブは周囲の状況に無関係にプロセスを止めるものであるが、プロセスがBLOCKを発行する時点に、状況が変化し、実際にはWait状態になる必要が無くなることもあり得る。この時にプロセスがBLOCKを発行してしまうと、その結果デッドロックになる。

これらの解決策としては、① BLOCKを行なう時点でその有無を調べ不用ならBLOCKを発行しない。② BLOCKは必ずかけるが、その時に何らかの方法で必ずプロセスが起動するように策を講じておくこと。が考えられる。これらの方法は、BLOCKと他に幾つかのプリミティブを組み合わせることによって実現することができる。具体的には図3-1のように示される。①に対するものとしてiii)が、②に対してはi), ii)がある。i)はSENDとBLOCKの間にRECEIVEを入れたもの。RECEIVEの相手は必ず受信するプロセスにする。簡便で使いやすい。この後でSENSEを出すことにより生じた通信の状態を知ることができる。ii)はTIMEとBLOCKとSENSEを組み合わせた形である。BLOCKの前にTIMEを発行するので必ず起動がかかる。iii)はSENSEを用いてBLOCKの可否を調べるものである。この場合は、どのプロセスもBLOCKの対策をする必要がある。

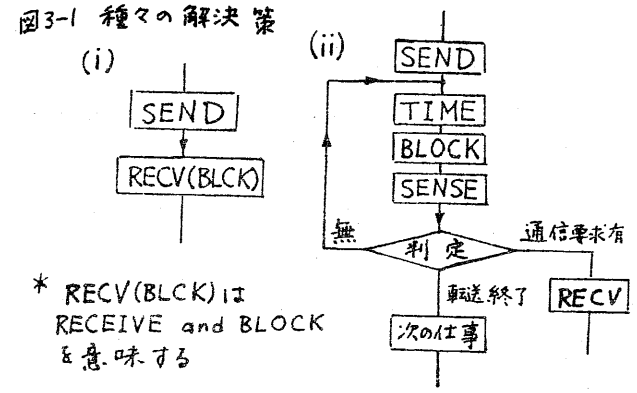
上記i)～iii)の他に解決策の①,②を兼ね備えたマクロとして、SENSE, BLOCK, TIMEを組合せ、且引数を用いたWAITを考えることもできる。これを用いれば、BLOCKの問題は起らない。

解決策の使用例 ―― i)を用いた場合

複数のプロセスと同時に通信をするプロセスMにおいては、通信の自由度が大きいので、自分がSEND(Lへ)-BLOCKを出した時点で相手プロセスLも同時にSEND(Mへ)-BLOCKを出すことがあり得る。

この場合Mは、SEND(Lへ)-RECEIVE ANY-BLOCKとすれば、デッドロックは、回避される。

図3-1 種々の解決策



＊ RECV(BLCK)は RECEIVE and BLOCK を意味する

WAIT 基本形

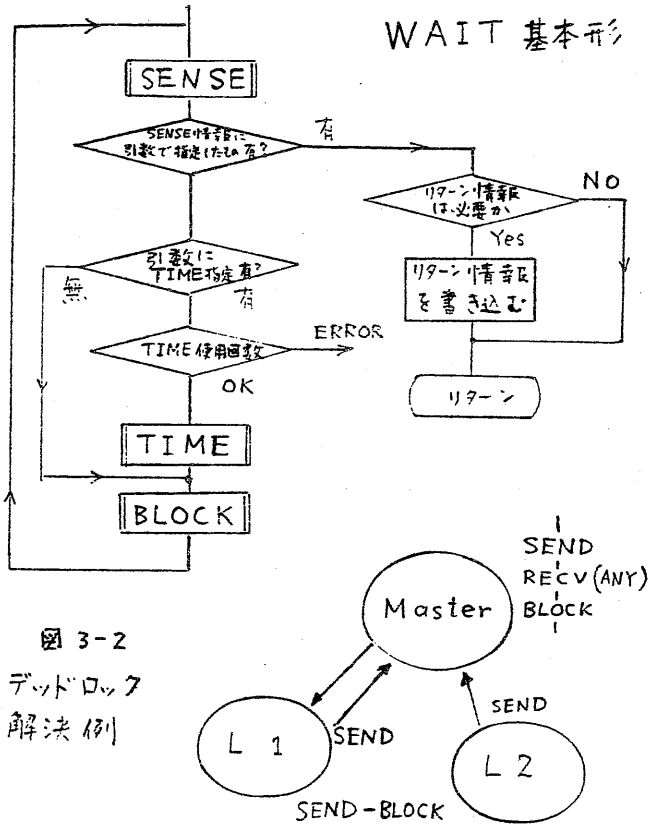


図 3-2
デッドロック
解決例

第2節　様々な問題点

前節における問題の他に実装上で気づいたことを、二・三考察する

I　新プリミティブの必要性

BLOCKの問題に対しては、前節で述べたようにWAITを設けることで解決できる。プロセス通信に関しては、通信を途中で打ち切るプリミティブ COMM.CANCELが実用上必要であろう。

II　実装の改善

プログラムを見直してみると、レジスタ類を活用すれば、プログラムを短かくできる。特にOKI版では、数回に渡る機能変更の為、実装能率が非常に悪くなっている。これを改善すると、3~4割 能率が向上する。

III　処理能率の向上

リモート通信の受信側で、メッセージパケットを受信した時のパケット合成法を、現在のような全パケットが揃ってから行なう方法から、到着パケット毎にプロセスの受信バッファ領域に書き込んでしまうやり方に変え、パケットの有効利用をはかる。通信制御プロセスレベルでのCloseコマンドの使い方を、Closeの合意を取りつける方法から、一方の通知のみで、止めることのできるようにして、能率向上を計る。

また、プロセス管理の面では、PSWやIWを核内から各ユーザプロセスの領域に移すこと、等の改善策が考えられている。

IV　核の改善

プロセス通信に関しては、実装済ではあるが、未使用であるリンクモードの充実をはかること。

異常時に対処するために、通信プリミティブにおける通信相手の指定を、現在のように単なるプロセス番号だけではなく、ポートも指定できるような機構を設け、ポートの違いによって異なるCSWを作成するようにすること。

通信のCANCELも出せなく、続行もできない状態の時に、核の方で、ガーベージコレクション等の方法により、打ち切ってやること。

核の機能としては、資源管理プログラムを強化して、I/Oを使い易くすること。等がある。

V　実装プリミティブの改善

現在実装の、ほとんどのプリミティブが、引数として、自分のプロセス番号を指定しなければならない。この点を改良して、自身のプロセス番号は書かないでプリミティブが使用できるようにしたい。プロテクションの一方法としても、これは重要であろう。特にSEND/RECVでは、現在の提出形式は引数の点で難があるので、これを使い易くして、ユーザ向きにしたプリミティブを作る必要がある。（詳細は次節で述べる）

第3節　ユーザ向きSEND/RECEIVE

ユーザ向き通信プリミティブは、各使用者毎に、プロセス名の変換テーブルを設けて、毎回のプリミティブ提出時に、使用者の決めた相手プロセス名に対するNameを用いるもので、Nameから実際のプロセス番号への変換は、このプリミティブ内で行なう。

使用目的　SEND/RECEIVEの相手の指定を簡単にすること。プリミティブ提出時に、通信相手の確認を行ない、許された相手のみと通信を行なうようにすること。

使用の前提条件　ユーザプロセス生成の時点で、PSW, IWやプロセス通信用のバッファを確保する。通信可能な相手プロセスを登録表に登録するプリミティブと、消去を行なうプリミティブを設ける。登録プリミティブでは、通信の可否を調べ、相手が否なら登録は行なわない。

提出法　各種の引数をテーブルに置き、このテーブルへのポインタを用いて、サブルーチンコール　を行なう。

手順　プロセスのCREATE時に作られた方針によって、登録プリミティブにより、相手プロセスの登録を行なう。ユーザ向きSEND／RECEIVE提出時に、登録表を参照して、通信の可否を調べる。否なら、エラーリターンに返す。

利点、　予め通信可能な相手を指定し、プリミティブ提出時に相手のチェックを行なうので、相手を誤って提出しても、核などの動作に異常は生じない。引数領域を別に設けたので、リエントラントに用いることができる。

第4節　まとめ

以上見てきたように、NOSは、計算機網に適するように、計算機の境界を意識しないでプロセス間通信を使用できることが大きな特徴である。また、これを階層構造によって構成したので、デバッグや保守が容易になり、拡張性に富んだものになった。

NOSの使用実験を通して、プリミティブを組み合わせた、使いやすいマクロの提案を行なった。

これからの課題としては異常時対策と評価の面がある。すなわち、異常時の対策が完全には組み込まれていないので、今後この方面の充実をはかる必要があること、また、プリミティブの実行所要時間やメッセージの伝達遅延の評価の問題が残っていること、等である。

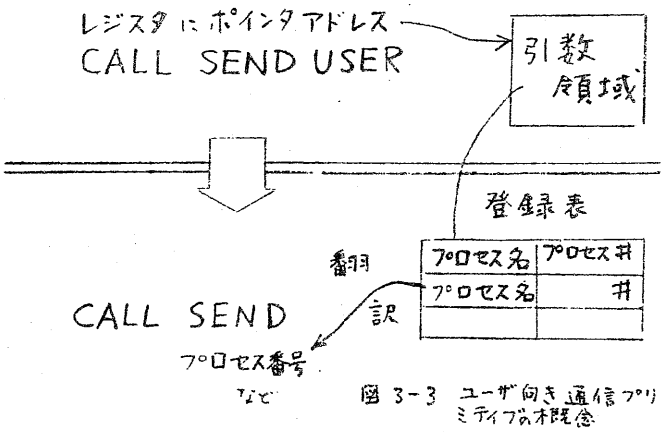


図 3-3　ユーザ向き通信プリミティブの概念

参考文献

Ⅰ　この論文の基になったものとして

① 田中・元岡「研究用電子計算機網 TECNET」EC 73-57

Ⅱ　NOSのPPS-1上の実装に関しては、

② 仲川明和「ポリプロセッサシステム管理用ファームウェア」東大工学系研究科 修論 1976

③ 山内・他「ポリプロセッサシステムPPS-1のオペレーティングシステム」信学研資昭和52年11月

Ⅲ　FACOM-Rに実装したものに関しては

④ 高橋篤哉「知能端末向きプロセス間通信」東大工,電子工学科 卒論 1976.

Ⅳ　NOSのジョブ管理に関しては

⑤ 小田,堀口 他「網向きジョブ管理システム」信学研資昭和52年11月

Ⅴ　網ファイル 管理に関しては、

⑥ 堀田敬志「コンピュータ・ネットワークにおけるファイルシステム」＊研究室内部資料1976

Ⅵ　OKITAC上に実装されたプロセス間通信システムの詳細については、

⑦ 田中・和賀井「NOSのプロセス間通信システム」　＊東大工電気工学科・元岡研 内部資料1977.3

付録　（用語の補注）

◆ローカル通信　通信を行なう2つのプロセスが同一計算機にある場合

◆リモート通信　通信を行なう2つのプロセスが異なる計算機にある場合