マイクロコンピュータ 23－1
（1982. 9. 17）

マルチプロセスモニタに基づく インテリジェント端末システム

重松 保弘　小出 真
（九州工業大学情報工学科）

1．まえがき

知能端末（intelligent terminal）は、比較的大規模のホスト計算機に接続される、比較的小規模の自立型端末であり、端末自身で、データの蓄積，加工，転送等を行なうことができる。非知能端末と知能端末の利用形態を図1に示す。知能端末の一般的な利用形態は（Ⅱ）であり、多くの場合、ファイル装置を有している。この場合の具体的な利用目的は、ホスト計算機の前処理（パスワード送出，セッション開始処理等），後処理（計算結果の簡易図形表示，印刷等），ファイル転送，ジョブ入力，出力検索[2]等である。知能端末は、ホスト計算機との接続関係においては端末であるが、それ自身は多くの場合、多目的に利用できるコンピュータシステムである。そこで、知能端末に通信システムを付加して相互接続させることにより、知能端末のネットワーク化を実現することができる（Ⅲ）。このような利用形態の利点としては、次の3点が挙げられる。

(1) 端末間通信がホスト計算機を介すことなく実現できる。そのため、ホスト計算機の通信制御負荷を軽減できると共に、効率的な端末間通信が可能となる。

(2) ホスト計算機のOSに手を加えることなく、システム全体の通信制御機能の拡張，及び、高度化がはかれる。

(3) ユーザープロセス間通信の機能を持たないホスト計算機においても、知能端末ネットワークを介して、容易にユーザープロセス間のメッセージ交換，及び同期処理が実現できる。

（Ⅲ）を複数ホスト計算機に拡張したシステムが（Ⅳ）である。このようなシステムでは、知能端末は特定のホスト計算機に所属しない。各ホスト計算機固有の端末プロトコルを選択することによって、知能端末はホスト計算機を選択し、その端末として動作する。また、端末プロトコル変換を行う知能端末（ゲートウェイ知能端末）を介して、ホスト計算機－ホスト計算機間通信を実現することも可能である。こうした知能端末ネットワークシステムの研究開発の最初の段階として、我々は、まず、マルチプロセスモニタを使用して（Ⅱ）タイプのシステムを作成した。マルチプロセスモニタを使用する主な利点は、多機能ソフトウェアのモジュール化が比較的容易に実現できることである。マルチプロセス化により、

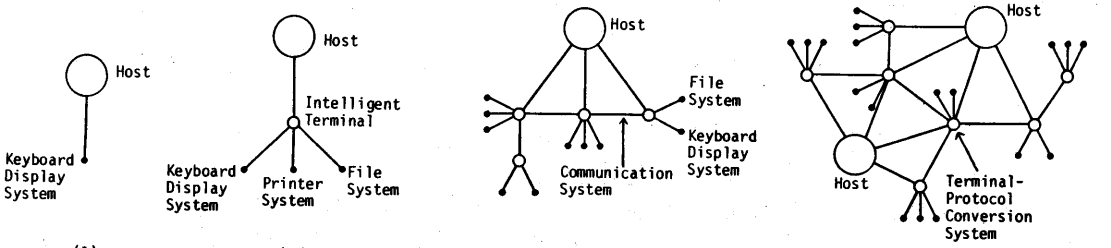


図1．非知能端末と知能端末の形態

(Ⅲ)，(Ⅳ)タイプのシステムへの拡張が容易になることも利点の1つである。本稿では、筆者らが実現した(Ⅱ)タイプのシステムについて述べる。

2．知能端末システムの構成

本知能端末システムは、図2に示すネットワークの一部として構成された。即ち、図2の破線部で囲まれた部分が本稿で述べるタイプ(Ⅱ)知能端末システムである。本知能端末は、筆者らによって開発されたマイクロコンピュータネットワークHOLENET[3]のホストマイクロコンピュータとしても使用される。また、この端末は、本学情報処理教育センターのMELCOM COSMO 700Ⅲに接続されているが、700Ⅲを通じて、本学情報処理施設のMELCOM COSMO 800Ⅲに接続することもできる。

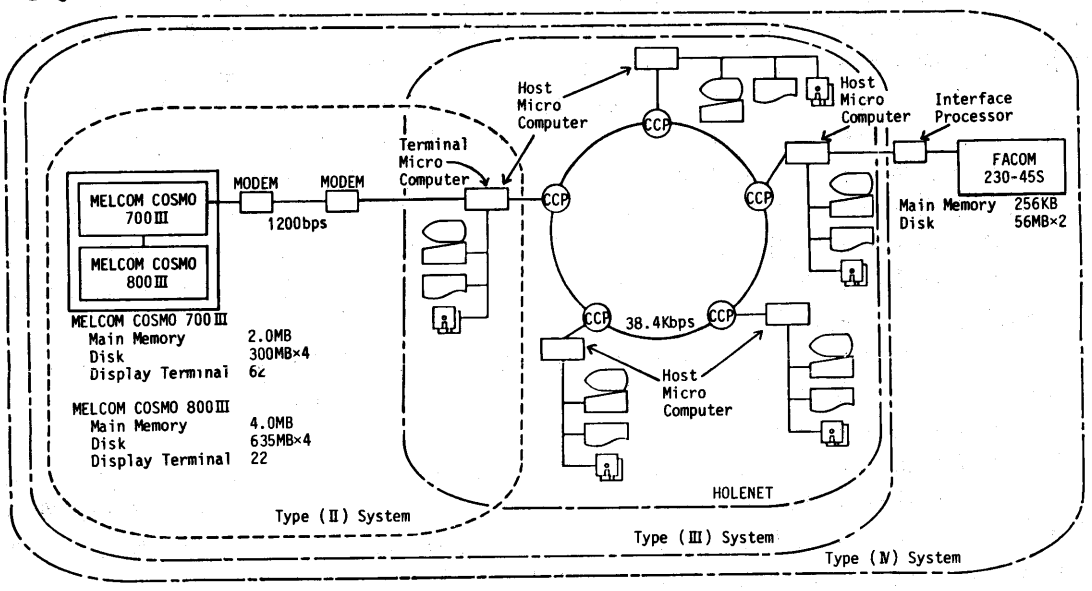


図2．知能端末システムの構成

3．知能端末システムの機能

知能端末システムに要求される主な特性，機能は、次の3点である。

(1) コマンドやデータの透過性が良いこと。即ち、非知能端末(Ⅰ)と知能端末(Ⅱ)が共存するホスト計算機システムにおいて、(Ⅰ)と(Ⅱ)の間でコマンドやデータの形式上の相異が少なく、(Ⅰ)と(Ⅱ)を交互に使用しても違和感が少ないことが要求される。

(2) ホスト計算機と知能端末間のデータフローパス(以下パスと呼ぶ)の設定，解放が容易であり、多様なパスの設定が行なえること。パスは、ホスト計算機と知能端末を接続する物理的な通信路の上に設定される論理的な通信路であり、基本的には、ホスト計算機上のプロセス $h_i \in H=\{h_1, h_2, \cdots\}$ と知能端末上のプロセス $t_j \in T=\{t_1, t_2 \cdots\}$ の順序対で表わすことができる。(但し、この場合のプロセスは、プログラムモジュールとしてのプロセスではなく、処理機能としての概念上

の実体を指している）　即ち、

$$p=(h_i,t_j)\in H\times T \quad \text{または} \quad p=(t_j,h_i)\in T\times H$$

(h_i,t_j)はホスト計算機から知能端末へのパス，(t_j,h_i)はその逆方向のパスを示す。もう1つのタイプのパスpは、分岐構造をもつものであり、1つのプロセスに複数のプロセスが接続される。即ち、

$$p=(h_i,\{t_\ell,t_m\cdots\})\in\{h_i\}\times 2^T \quad \text{または} \quad p=(\{h_\ell,h_m,\cdots\},t_j)\in 2^H\times\{t_j\}$$

これらのパスが、1つの物理的通信路の上に複数本設定されている状態を、パス多重状態　と呼ぶ。タイプ（Ⅲ），タイプ（Ⅳ）の知能端末システムを実現するには、これらのシステム上でパス多重状態が実現できることが望ましい。しかし、現在のほとんどのホスト計算機はこれを許さないので、我々のシステムでは、分岐構造を持つパスを実現することにした。また、以下、知能端末上のプロセスとしては、I/Oデバイスを制御するプロセスのみを扱う。即ち　$T=\{console,file,printer\}$である。図3に、本知能端末システムにおけるパスの分岐構造と，パスの設定，解放コマンドによるこれらの構造間の遷移を示す。

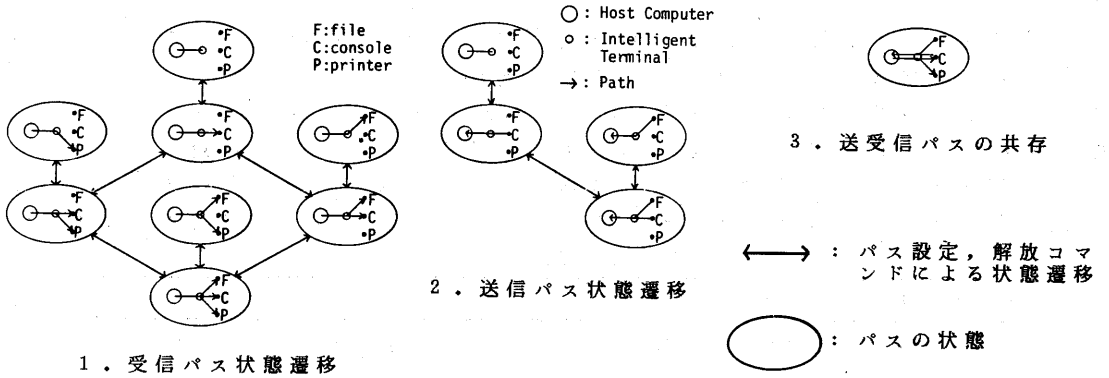


図3．パスの分岐構造と状態遷移

(3)　端末動作モードと、自立動作モードの共存が可能であること。即ち、ホスト計算機の端末としての動作を行いながら、知能端末上で、これとは独立した動作を行えることが要求される。この典型的な利用形態は、ホスト計算機からの受信データを低速プリンター装置に出力しながら、これと並行して、利用者が知能端末中のファイルの編集，翻訳処理等を行なうものである。本システムでは、(2)で示したパスの分岐構造において、consoleがパスに含まれていない状態、即ち、$\{p\mid p=(h_i,t_j),t_j\in T,console\notin t_j\}$にある時、利用者はコンソールからユーザープロセスの起動を試みることができる。

以上の機能を実現するために必要な機能モジュールと、各モジュール間のデータの流れを図4に示す。各モジュールは、割り込みによって駆動されるルーチンまたは、相互に並行動作可能なプロセスであり、各々、次のような機能を持つ。

(1)　受信ルーチン（Receiver Routine）：　ホスト計算機から送られてきたデータをを受信し、受信バッファに蓄積する。（割り込み駆動ルーチン）

(2)　出力管理プロセス（Output Control Process）：　受信バッファ中のデータを端末

のコードに変換した後、受信パスが設定されているプロセスへ送る。

(3) 端末制御プロセス（Terminal Control Process）：　キーボードから入力されたデータが知能端末のコマンド（表1）であるかどうかを検査し、そうであれば、その機能に従ってパスの設定，解放等の処理を行う。また、キーボードから入力されたデータが表1で示すコマンドでなければ、送信プロセスへそのデータを送る。

(4) 送信プロセス（Transmitter Process）：　ホスト計算機へ送信バッファ中のデータを送信する。

(5) ディスプレイ出力プロセス（Display Process）：　ディスプレイ装置へデータを出力する。

(6) プリンター出力プロセス（Printer Output Process）：　プリンター装置へデータを出力する

(7) ファイル出力プロセス（File Output Process）：　フロッピーディスクのファイルにデータを出力する。

(8) ファイル入力プロセス（File Input Process）：　フロッピーディスクのファイルの内容を送信プロセスへ送る。

(9) キーボード入力プロセス（Keyin Process）：　キーボードからデータを入力する。

(10) 受信バッファ管理プロセス（Receive Buffer Control Process）：　受信バッファを監視し、受信バッファのオーバーフローを防ぐ。

(11) メッセージサービスプロセス（Message Service Process）：　I/Oデバイスの状態を表わすメッセージを、必要に応じてディスプレイプロセスへ送る。

なお、(6)～(8)の、ディスプレイ装置及びキーボード装置を除くI/O装置に対する入出力を制御するプロセスを総称して、デバイスコントロールプロセスと呼ぶ。

表1．知能端末のコマンド

パス設定，解放コマンド	（^　－－－　コントロール）
^P	プリンター装置への受信パスの設定（80文字/1行で印字）
^L	〃　〃　（132文字/1行で印字）
^O	〃　受信パスの解放
^W	フロッピーディスク上のファイルへの受信パスの設定
^E	〃　受信パスの解放
^R	フロッピーディスク上のファイルからの送信パスの設定
^C	〃　送信パスの解放
^D	コンソールに対する送受信パスの解放
^S	〃　〃　（偽受信パスの設定）
^¥	〃　〃　（ブレークキャラクター送信）

端末制御コマンド	
^A	英文字の大文字，小文字の切り換え
^U	パスの設定，解放、及び、I/O機器の状態を表示

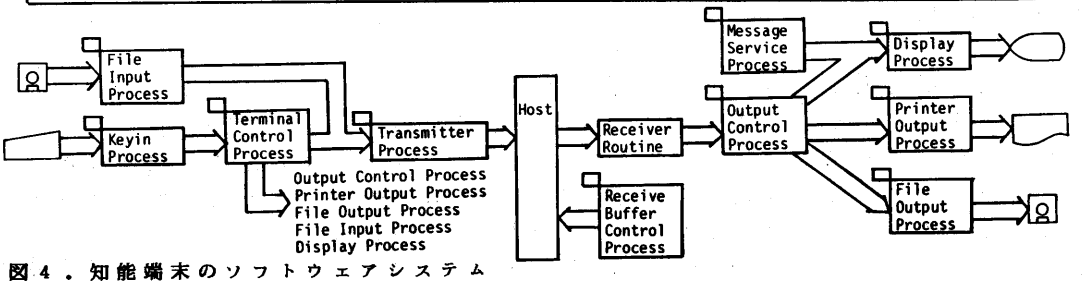


図4．知能端末のソフトウェアシステム

4．マルチプロセス化による問題とその解決法

ここでは．3で述べた機能を実現するために解決しなければならないいくつかの問題点と、その解決法について述べる。

4.1　通信、同期、相互排斥の手段

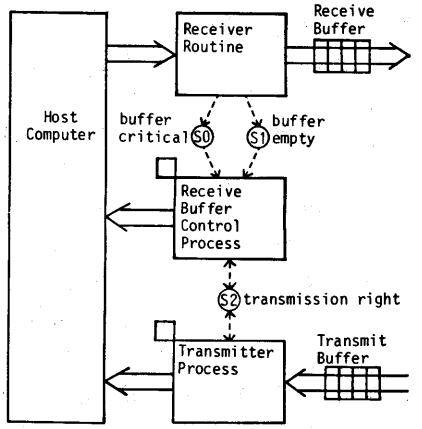
マルチプロセスモニタには、通常、各機能モジュール間で、同期，通信、相互排斥を行なう手段が用意されている。筆者らが採用したMP/Mモニタでは、次に示す様に、フラグ管理，キュー管理の機能が、この為に用意されている。

〔ＭＰ／Ｍファンクション〕

- ○　同期の手段
  - フラグ set　ファンクション
  - フラグ wait ファンクション
- ○　通信、及び、相互排斥の手段
  - キュー write ファンクション
  - キュー read　ファンクション

4.2　送信権の相互排斥

ホスト計算機から端末にメッセージを送信する場合、端末中の受信バッファのあふれによる受信オーバーランの発生を防ぐ為、ホスト計算機と端末は、通信線上のデータの流量を制御しなければならない。これを解決する1つの方法は、端末が受信バッファの状態を常時監視しておき、受信バッファの空領域が少なくなった時ホスト計算機に制御シーケンスを送付して、送信の一時停止を要求する方法であり、本システムのホスト計算機はこの方法を採用している。この方法を用いる時必要となる相互排斥の対象は、ホスト計算機への送信権である。送信権は、図4における受信バッファ管理プロセスと送信プロセスの間で相互排斥される。この2つのプロセスと受信ルーチンと、その処理の概要を2値セマフォ（semaphore）を用いて記述したものを、図5，図6に示す。ただし、3個のセマフォ変数のうち、S0，S1の初期値は0，S2の初期値は1である。またsignalの初期値はfalseである。



S0 : buffer critical
（バッファの空領域が少ない）

S1 : buffer empty
（バッファが空である）

S2 : transmission right
（送信権は解放されている）

（　）内は各セマフォ変数が1の場合の意味

図５．送信権の相互排斥

4.3　端末の透過性による問題

端末上のコンソールからホスト計算機への制御シーケンスを送付できるような、コマンドの透過性を持つシステムでは、図6の送信プロセスを改めなければならない場合がある。本システムのホスト計算機は、図6からわかるように、送信の一時停止を求める制御シーケンスとして、2文字のエスケープシーケンスを使用

し（ESC,'H'）、送信の再開にはESCコード／文字のみを使用している。送信バッファ中にコンソールからキー入力されたエスケープシーケンス（ESC,'H'）が入っている場合を考えると、次の手順で送信権が移動した時、問題が発生する。

step 1 ：送信プロセスが送信権を獲得し、ESCコードを送信し、送信権を返す
step 2 ：受信バッファ管理プロセスが送信権を獲得し、（ESC,'H'）を送信し、さらに受信バッファが空になった後、ESCコードを送信し、送信権を返す
step 3 ：送信プロセスが、送信権を獲得し、'H'を送信し、送信権を返す

この場合、少なくとも、コンソールから入力された送信一時停止コマンド（ESC,'H'）は破壊される。我々のシステムでは、連続する2個のESCコードを受信したホスト計算機はブレーク（break）動作に移り、続く'H'を受信した後、（ESC,'H'）で送信を一時停止することになる。この問題の1つの解決策は、エスケープシーケンスを一括して送ることができるよう、送信プロセスを改良することである。ただし、受信バッファのオーバーフローを招かないよう、エスケープシーケンスの2文字が共に送信バッファに存在する時のみ、送信権を要求しなければならない（図7）。しかし、図7の送信プロセスを使用したとしても、なお、次の手順では問題が発生する。

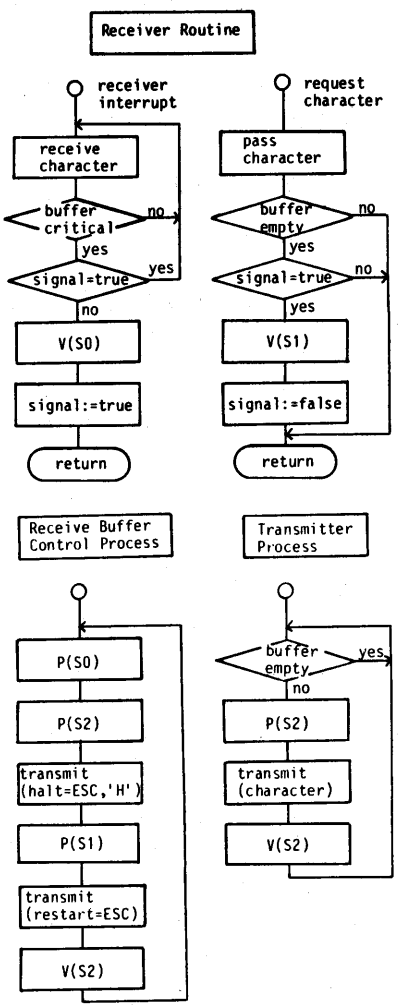
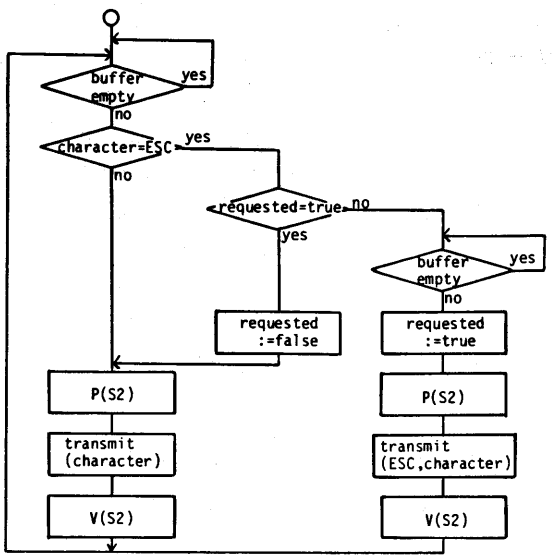


図6．図5のプロセスとルーチンの処理手順

step 1 ：送信プロセスが（ESC,'H'）を送信
step 2 ：受信バッファ管理プロセスが（ESC,'H'）を送信、しばらくしてESCコードを送信
step 3 ：送信プロセスがESCコードを送信

この場合、step 1で送信を一時停止したホスト計算機は、step 2の最初のESCコードで送信を再開し、'H'を受信した後、連続する2文字のESCコードでブレーク動作に移る。この問題を避けるためには、ホスト計算機からの送信を一時停止させる権利の相互排斥を実現しなければならない。これは、送信プロセスを更に図8に示すように改良して達成される。なお、本システムのホスト計算機で採用されている制御シーケンスには、送信の一時停止を要求する（ESC,'H'）を含め、タブシミュレーション，ブレーク等、36種のエスケープシーケンスが含まれてい



初期値
requested : false

図７．エスケープシーケンスのための送信プロセス

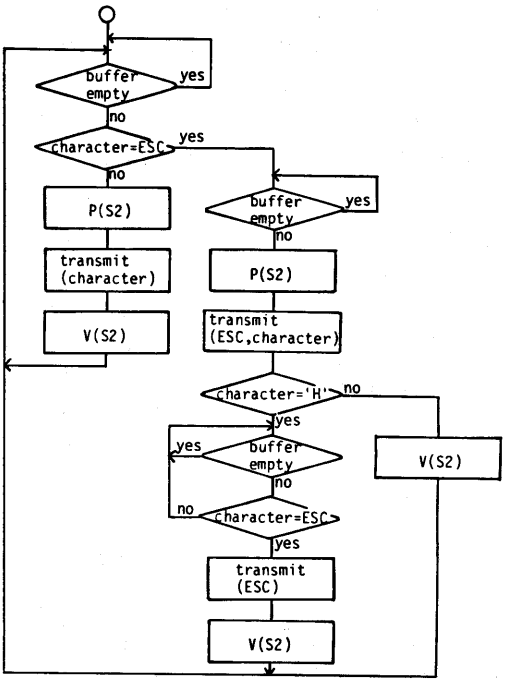


図８．送信一時停止コマンドのための送信プロセス

る。図8では、これらエスケープシーケンスを送信する場合の動作も含めて記述している。

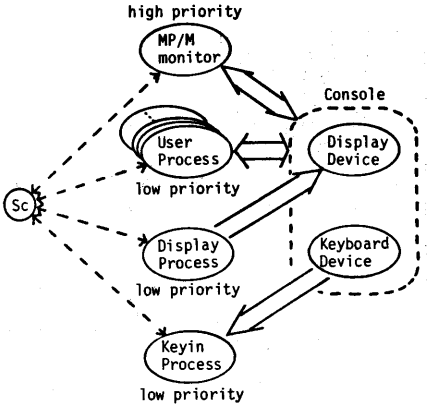
4.4 コンソール使用権の相互排斥

通常、モニタは、ディスプレイ装置とキーボード装置を1つのコンソールとして取り扱う。したがって、モニタファンクションを利用するユーザープロセスも、コンソールを一体として使用することになる。しかし、本システムのように、ディスプレイ装置への表示要求とキー入力が非同期的に起こるシステムでは、ディスプレイ出力とキー入力を行うプロセスは独立に与えられなければならない。それは、これらを1つのプロセスにすると、キー入力の監視、バッファの監視、あるいは、プロセスのディスパッチ(dispatch)を頻繁に行わねばならず、システム全体の効率が悪化するからである。この時、コンソールは、この2つのプロセスの他に、複数のユーザープロセス、及び、MP/M モニタによっても共用されることになり、これらの間で、コンソールの使用権を相互排斥する必要が生じる。コンソールの使用権を2値セマフォ変数Scを用いて相互排斥するシステムを図9に示す。しかし、図9に示すような単純なシステムでは、次のような問題が発生する。

(1) ディスプレイ出力中にキー入力できない
(2) キー入力中にディスプレイ出力ができない
(3) モニタ以外のプロセスがコンソール使用権を手放すと、コンソール使用権に関して優先順位の高いMP/Mモニタに常にコンソール使用権を横取りされる。その

ため、ディスプレイ出力プロセスとキーボード入力プロセスの間で、交互にコンソール使用権をたらい廻しできない。

1つの解決策を図10に示す。すなわち、コンソールへのパスが設定された時は、キーボード入力プロセスがコンソール使用権を代表して獲得し(P(Sc))、ディスプレイ出力プロセスにこれを通知して(V(Sd))、ディスプレイ出力プロセスとキーボード入力プロセスでコンソールを共用する方法である。入力コマンドの指示でコンソールをパスから解放する時、キーボード入力プロセスは、ディスプレイ出力プロセスからコンソールの使用権を取り上げた後(P(Sd))、これを返却する(V(Sc))。



初期値 Sc : 1

図9．コンソール使用権の相互排斥

4.5 I/Oデバイスへの出力権の相互排斥

端末制御プロセスは、キーボード入力プロセスから渡されたコマンドを解析した後、応答メッセージをディスプレイ出力プロセスに送る。また、メッセージサービスプロセスは、デバイスの状態変化をデバイスコントロールプロセスから受

Sc：コンソール使用権（1）
Sd：ディスプレイ出力権（0）
Sp_0：ディスプレイ装置に対するパス表のアクセス権（1）
Sm_0：ディスプレイ出力プロセスへのデータ送信権（1）
Si：I/Oデバイス使用権（1）
Spi：I/Oデバイスiに対するパス表のアクセス権（1）
Smi：I/Oデバイスiへのデータ送信権（1）

（ ）内は初期値

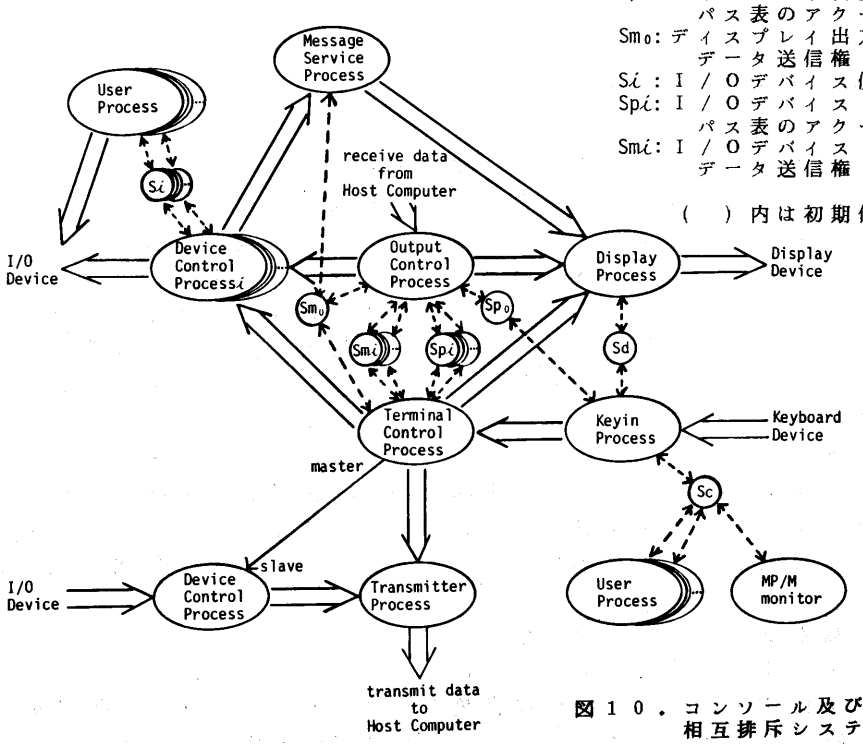


図10．コンソール及びI/Oデバイスの相互排斥システム

け取り、メッセージをディスプレイ出力プロセスに送る。そこで、出力管理、端末制御、メッセージサービスの各プロセスの間で、ディスプレイ出力プロセスへのデータの送信権を相互排斥しなければならない。このために、図10のセマフォ変数S_{mo}が使用される。また、端末制御プロセスは、キー入力されたコントロールコード(表1)に従ってパスの設定、解放を行う時、適当なデバイスコントロールプロセスに対し制御データを送り、その指示を行っている。そのため、ディスプレイ出力プロセスへのデータの送信権の相互排斥の場合と同様に、デバイスコントロールプロセスiへのデータの送信権を相互排斥する目的で、図10のセマフォ変数S_{mi}が使用される。

4.6 受信パス表の相互排斥

プロセス間において、受信パスの設定,解放の通知は、受信パス表を介して行われる。そこで、受信パス表の各エントリを相互排斥するために、2値セマフォ変数($S_{p0}, S_{p1}, S_{p2}, \cdots$)を使用する。受信パス表を利用した具体的な受信パスの設定,解放の手順を、コンソールへのパスの設定,解放を例にとって述べる。

コンソールの使用権を獲得したキーボード入力プロセスは、受信パス表を獲得し($P(S_{p0})$)、これに1を記入した後手放す($V(S_{p0})$)。また、キーボード入力プロセスは、図11の手順でコンソールへのパスを解放し、コンソールの使用権を手放す。また、出力管理プロセスは、図12の手順で受信パス表を検査しながらディスプレイ出力プロセスへデータを送る。他のI/Oデバイスに対するパスの設定,解放も、同様に行われる。

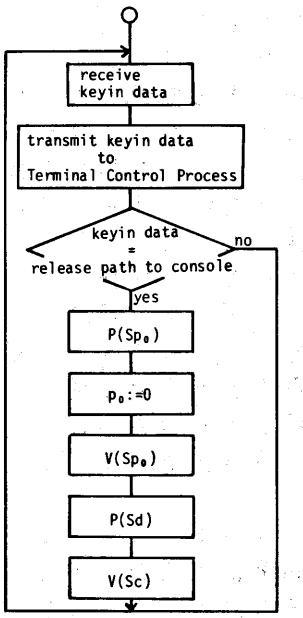


図11. キーボード入力プロセスのコンソール使用権の解放手順

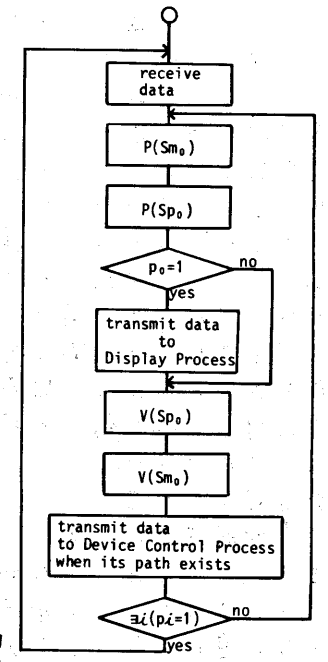


図12. 出力管理プロセスのディスプレイ装置へのデータ出力手順

なお、P操作,V操作によるオーバーヘッドが無視できない場合、受信パス表の全エントリーの獲得およびデバイスコントロールプロセスへの送信権を示す2値セマフォを用意し、パスの存在の確認、また、存在するパスに対するデバイスコントロールプロセスへのデータの送信を一括して行う方法も考えられ、本システムは、この方法を採用している。

4.7 送信パスの制御

ホスト計算機へのデータの送信は、端末制御プロセスを主プロセス、デバイスコントロールプロセスを従プロセスとする主従関係で制御する。このため、送信

パス表は使用しない。また、分岐構造を持つ送信パスが決定されていても、実際には、上述の2つのプロセスのいずれかのみが送信権を有し、送信プロセスへデータを送ることができる。

5. あとがき

知能端末のソフトウェアシステムをマルチプロセス化することにより、自由度の多い、また、信頼性の高いシステムを開発することができた

先に述べたように 知能端末のソフトウェアシステムをマルチプロセス化することにより、数々の利点を得ることができるが、実際のシステムを作成する場合、システムの効率を上げるため、次のようないくつかの考慮が必要となる。

(1) プロセスのディスパッチを意識したプログラミングを行わなければならない場合がある。例えば、あるプロセスがウェイトしないで状態待ちを行う時、その状態は、他のプロセスによって引き起こされるものとしよう。この場合、状態待ちをしているプロセスが無意味なチェックのくりかえしを行うのを防ぐために、1度チェックを行った後、プロセスのディスパッチを行うようにする必要がある。

(2) モニタに用意されている、P操作、V操作に関するファンクションのオーバーヘッドが大きい場合、共通メモリ領域、及び、モニタに用意されているプロセスディスパッチに関するファンクションを利用して、簡単なP操作、V操作を実現することにより、効率を上げられる場合がある。ただし、共通メモリのアクセスに対し、排他制御を成立させるための十分な考慮が必要である。本システムでは、プロセスディスパッチのためのタイマー割り込みを禁止することで、排他制御を行っている。

本知能端末は、8ビットのマイクロプロセッサを使用しているが、マルチプロセス化によっても十分効率よく使用できることを確認した。さらに16ビットのマイクロプロセッサが利用されるようになると、知能端末のマルチプロセス化は、さらに有効になると考えられる。

謝辞

本システムの作成にあたり御協力いただいた、本学情報処理教育センターの矢嶋虎夫助教授、中山泰雄助手、中村為雄助手に感謝します。

参考文献

1) P. Brinch Hansen : Operating System Principles (1973)
2) 野上, 重松 : マイクロコンピュータを用いた出力検索システム : FAMOUS 1 情報処理学会論文誌, Vol.21, No.2, pp.167~170 (1980)
3) 重松, 柴田, 小出 : 教育・研究用マイクロコンピュータネットワーク : HOLENET 情報処理学会分散処理システム研究会資料 (1982)
4) MP/M USER'S GUIDE , DIGITAL RESEARCH
5) MELCOM UTS/VS タイムシェアリング説明書, 三菱電機