マイクロコンピュータ 40—2
（１９８６．６．２７）

ホスト・コンピュータ上でのマルチタスク処理を支援するマルチウィンドウ端末の実現

清水謙多郎，石田晴久
東京大学大型計算機センター

本稿では、ワークステーション上でホスト・コンピュータのマルチウィンドウ端末を実現し、ホスト・コンピュータ上でのマルチタスク処理を支援する通信ソフトウェア・システムＷＩＰについて述べる。ＷＩＰは、ホスト・コンピュータとワークステーションの間の通信回線を多重化して用いることにより、複数の端末セッションおよびファイル転送を並行して行うことを可能にしている。ＷＩＰはユーザ・レベルのプログラムとして実現され、ワークステーション側のソフトウェアは現在ＯＳの異なる３機種のワークステーション上で稼動中である。本稿では、また、ＷＩＰのようなシステムを実現する上で既存のＯＳに必要とされる機能等に関し考察を行う。

Implementation of multiwindow terminals for supporting multitasking on host computers

Kentaro Shimizu and Haruhisa Ishida
Computer Centre, University of Tokyo

This paper describes the communication program WIP (Window Interface Program) which implements multiwindow terminals on workstations to support multitasking on host computers. WIP allows a user to interact with multiple command interpreters of a host computer and transfer files concurrently by multiplexing the communication line between the host and the workstation. Versions of WIP are implemented as a user-level program and are now running on workstations in three different operating systems. This paper also discusses the operating system facilities needed to implement WIP-like system as a user-level program.

ホスト・コンピュータ上でのマルチタスク処理を支援するマルチウィンドウ端末の実現

清水謙多郎　　石田晴久
（東京大学大型計算機センター）

1．まえがき

近年、マルチウィンドウ機能は、Lispマシンや専用のＯＳを搭載した高性能のワークステーションのみならず、Concurrent CP/MやMS-Windowsにみられるように一般のパーソナル・コンピュータでも利用できるようになってきた。高解像度ディスプレイやポインティング・デバイスとしてのマウスを使ったマルチウィンドウ機能は優れたマン・マシン・インタフェースを提供し、すでにＯＡ、ＣＡＤ、プログラム開発などの分野で幅広く利用されている。

こうしたマルチウィンドウ機能を備えたワークステーションあるいはパーソナル・コンピュータの新しい利用法として、ホスト・コンピュータの操作性の向上を目的としたマルチウィンドウ端末としての利用が考えられる。すなわち、単にホスト・コンピュータ上の処理とワークステーションあるいはパーソナル・コンピュータ（以下では、ホスト・コンピュータに対する利用形態に着目し、両者を区別せずにワークステーションと呼ぶことにする）上のローカルな処理を並行して実行するだけでなく、ホスト・コンピュータのマルチタスキング機能を積極的に利用し、その様子をワークステーション上で、マルチウィンドウにより表示させるというものである。この場合、ウィンドウの管理、ウィンドウの表示のためのハードウェアおよびソフトウェア資源はワークステーション上のものを用いるので、分散処理の立場からホスト・コンピュータの負荷を削減することができると同時にホスト・コンピュータだけでは効率等の面で実現困難な優れたマン・マシン・インタフェースの利用が可能となる。

本研究はこのようなマルチウィンドウ端末を実現するシステムＷＩＰ（Window Interface Program）の開発とその評価に関するものである。

我々がマルチウィンドウ端末と呼ぶ機能はすでに商品化されているマルチセッション機能〔1〕と異なり、特殊なハードウェアを用いることなく（少なくとも１本の端末回線で）、ホスト・コンピュータに対する１つのセッションの中でマルチタスキングを行う。従ってマルチセッション機能と組み合わせて利用することも原理的に可能である。また、従来マルチウィンドウ端末を実現したシステムとして、R. Pike がグラフィック・ターミナル Blit上にウィンドウ管理や通信のためのソフトウェアをインストールし実現したシステム〔2〕があるが、本研究では新たな試みとして、

（1）　ワークステーション上の既存のＯＳに利用者レベルのプログラムとして容易にインプリメントできるシステムの開発を目的とし、実際に表１に示す３つのワークステーション上でインプリメントを行ってその容易性を実証した。

（2）　ホスト・コンピュータとワークステーションとの間で通常の対話セッションとファイル転送を並行して行えるようにし、さらにファイル転送自身の多重化を実現してその効果を測定した。

マシン	ＯＳ	記述言語	プログラム・サイズ
NEC PC9801	Concurrent CP/M	C,8086アセンブリ言語	1,030 行
Hitachi 2050	HI-UX	C	950 行*
Xerox 1100SIP	Interlisp-D	Interlisp-D	570 行
Vax-8600	Ultrix-32	C	800 行

上段：　ワークステーション
下段：　ホスト・コンピュータ
*拡張機能をもつがその部分は除く

表１．実現したシステム

（3） ワークステーション、ホスト・コンピュータ双方において、アプリケーションに応じたタスクを自由にＷＩＰに組込んで動かせるようインタフェースを整え拡張性をもたせた。

（4） この種のシステムを利用者プログラム・レベルでインプリメントする上で既存のＯＳに必要とされる機能および特に望ましい機能について、実際にインプリメントした経験を交えて考察を行った。

以下、２章ではＷＩＰシステムの基本構成について述べ、３章，４章および５章ではそれぞれ、ＷＩＰにおけるメッセージの転送方式、ファイル転送を中心とした拡張機能、およびＷＩＰのインプリメンテーションについて比較的詳細な議論を行う。また、６章ではＷＩＰの利用者インタフェースについて簡単に説明する。

２．システムの基本構成

図１はＷＩＰによるホスト・コンピュータとワークステーションとの結合を示したものである。ホスト・コンピュータ上で起動されるタスク i とワークステーション上に生成されるウィンドウ i とがそれぞれ１対１に対応し、利用者はワークステーション上のウィンドウ i を介して、ホスト・コンピュータのタスク i と交信する。すなわち、ホスト・コンピュータのタスクi に対応して仮想端末i が生成され、仮想端末i から入力したデータはタスクi の入力データとなり、タスクi の出力データは仮想端末i に送られる。この際、仮想端末 i の内容がワークステーションのディスプレイ上のウィンドウ i に表示される。ホスト・コンピュータとワークステーションの間は物理的に１本の回線で結ばれているだけなので、回線を多重化するためパケット形式でメッセージのやりとりを行う。各パケットは、メッセージがワークステーションのどのウィンドウに対応するものか、すなわちホスト・コンピュータのどのタスクに対応するものかを識別するためのフィールド（widフィールド）をもつ。

ＷＩＰはホスト・コンピュータ、ワークステーションの双方で稼動する。以下、ホスト・コンピュータ側のＷＩＰをリモートＷＩＰ、ワークステーション側のＷＩＰをローカルＷＩＰと呼んで区別することにする。

リモートＷＩＰはワークステーション側から送られてくるメッセージ・パケットを受け取り、widフィールドを調べ、対応するタスクにメッセージ・データを与える。また、 n 個のタスクが出力するデータはそれぞれのタスクに対応するウィンドウの識別番号をwid フィールドに入れ、パケット化してワークステーション側に送出する。一方、ワークステーション上のローカルＷＩＰはリモートＷＩＰとほぼ対称的な構成で、ホスト・コンピュータより送られてきたパケットを切り換え、指定されたウィンドウにデータを表示するとともに、現在のウィンドウから入力されたデータをパケット化してホスト・コンピュータに送出する。

我々はリモートＷＩＰとしてUltrix（4.2 bsd Unix相当）を搭載したVAX8600、ローカルＷＩＰとして表１に示す３つのワークステーション上でインプリメントを行った。

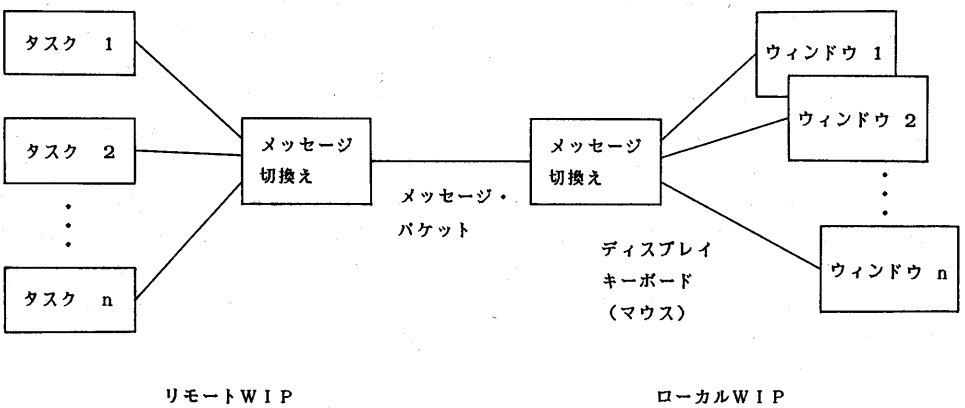
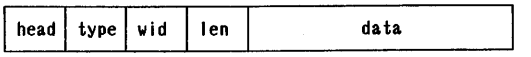
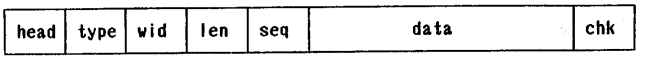


図１．ＷＩＰシステムの概略

(i) 通信パケット



(ii) ファイル転送パケット



head : ヘッダ
type : タイプ
wid : ウィンドウ識別子
len : パケット長
data : データ
seq : シーケンス番号
chk : チェックサム

図２．メッセージ・パケットの形式

名称	タイプ	意味
GEN	G	タスクの生成要求
KILL	K	タスクの消去要求
QUIT	Q	ＷＩＰセッションの終了要求
SET	S	各種モードの設定要求
ACK	Y	上記の各要求に対する肯定応答
NAK	N	上記の各要求に対する否定応答
DATA	D	通常のメッセージ・データ

表２．通信パケットのタイプ

３．メッセージの転送方式

３．１ パケットの形式

ホスト・コンピュータとワークステーションの間でやりとりされるパケットの形式は通常の対話に用いられる通信パケットとファイル転送時に用いられるファイル転送パケットの２種類がある。図２にパケットの形式を示す。

すべてのパケットは上に述べた回線多重化のためのwidフィールドの外、head、type、len、およびdataの各フィールドをもつ。文字コードとしてはASCIIコードを用いている。headフィールドはパケット転送の際、同期をとるためのもので、通常Control-A（SOH）が充てられる。 typeはパケット・タイプを識別する文字データ、lenはパケットの長さ（byte数）が指定される。

dataフィールドには転送すべきデータそのものが格納される。

ファイル転送パケットは通信パケットの各フィールドにシーケンス番号（seq）とチェックサム（chk）を追加した形式をとっている。これはまた、現在計算機間のファイル転送用システムとして広く用いられているKermit〔3〕のパケット形式にwidフィールドを追加した形でもある。ファイル転送の詳細については４章で述べる。

パケットはheadフィールド以外、ASCIIコードの印字可能文字から構成される。wid、lenおよびファイル転送パケットのseq、chkの各フィールドは、本来の整数値に空白文字コード（32）を加え、 ASCIIコードの印字可能文字に変換して転送する。また、dataフィールド中の非印字可能文字は、その文字コードの第７bitを反転して印字可能文字に変換するとともに引用文字（＃）を前に置き ２byte化して転送する。この方式はKermitの方式と同じで、非印字可能文字が機種によって特別に扱われるのを防ぐことを目的としている。

３．２ 通信プロトコルと性能

表２はＷＩＰの通信パケットのタイプを記したものである。

ローカルＷＩＰからリモートＷＩＰに対しタスクの

生成（GEN）、タスクの消滅（KILL）、各種モードの設定（SET）、およびＷＩＰセッションの終了（QUIT）が指示される。リモートＷＩＰはこれらの要求に対し、ローカルＷＩＰにACK、NAKのいずれかを返して要求が受け付けられたか否かを通知する。通常のメッセージ・データはDATAパケットにより転送されるが、この場合は応答性を考慮してACK／NAKによるハンドシェイクは行っていない。ただしＷＩＰでは常にタイムアウトを監視しており、通信がデッドロックに陥ることを防いでいる。

GENパケットのdataフィールドには新たなタスクとして実行すべきプログラムの名前を指定することができる。通常の対話セッションではコマンド・インタプリタが指定され、ファイル転送時にはkermitのサーバ・モードに相当するファイル転送専用のプログラムが指定される。

パケット化に伴うオーバヘッドは応答性に大きな影響を及ぼす。通常の対話セッションにおいて転送される文字数のオーバヘッドPは、１パケット当たりの制御用フィールドの文字数が４であることから、次のような式で表される。

$$P=\{4+q(l-4)\}/l$$

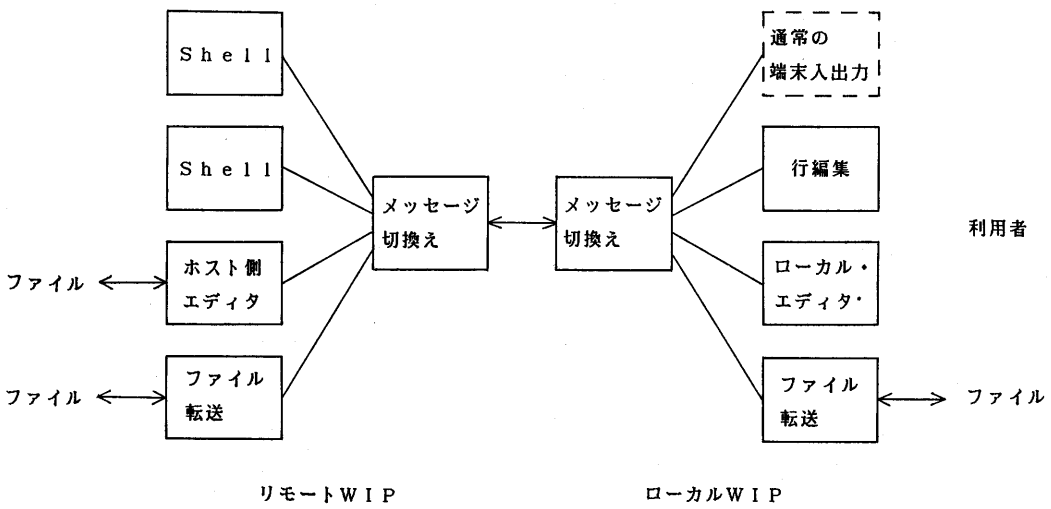
ここに、lはパケット長の平均値、qは全データ文字数に対する非印字可能文字の引用によって増加する文字数の比である。

実際にlおよびqの値を測定したところ、ワークステーション側で入力された文字データを１文字単位でホスト・コンピュータに送りホスト・コンピュータによるエコー・バックをそのまま利用したとき、l=13.8、q=0.54となり、P=0.33であった。一方、ワークステーション側でローカル・エコーを行い行単位の入力を許したとき（ホスト・コンピュータによるエコー・バックは行わない）、l=34.5、q=0.050となり、P=0.11であった。これらの値から、行単位の入力を行うことにより、転送される文字数のオーバヘッドは半分以下に減ることが分かる。その上、前者の場合メッセージ切換えのための時間的オーバヘッドが増大するので、実際の効果はさらに大きいものとなろう。

上記のようなパケット化を行わずに、通常は文字単位でデータを転送し、タスクが切り換わったときのみ、それを告げる制御情報を転送するという方法も考えられる。この場合、転送される文字数のオーバヘッドPは次の式で与えられる。

$$P=(q+r+s)/(1+q+r+s)$$

ここに、rはタスクの切換えを通知するのに必要な文字数の全データ文字数に対する比、sはその情報を通常の文字データと区別するのに使用する文字の引用に必要な文字数の全データ文字数に対する比とする。rおよびsは実際に極めて小さい（小さくすることができる）ので効率は先程の場合に比べかなり改善される。しかしながら、我々は後述するファイル転送などのアプリケーションとの整合性や拡張性を考慮し、上記のようなパケット転送方式を採用した。



現在検討中

図３．アプリケーションに応じたタスク間通信

4．拡張機能

4．1 拡張機能の組込み

ＷＩＰの拡張機能として、現在ファイル転送機能およびホスト・システム（Ultrix）がサポートする入力行編集をローカルに行う機能が組み込まれている。これらの機能はメッセージ切換えタスクとは別の専用のタスクとして実行される。これは、（1）両者を並行して実行することにより、他のウィンドウ―タスク間通信の応答性に与える影響を低減する、（2）アプリケーションに依存した処理を１つの独立したプログラムとして記述することで、システムのモジュール化を促進するとともに拡張性を高めるためである。図３はアプリケーションに応じたタスク間通信の様子を示す。分散エディタは将来の拡張として考えている。

アプリケーション依存の情報はすべて通信パケットのdataフィールドに格納される。我々はkermitと同様の転送パケット形式、転送プロトコルでファイル転送を行うプログラムを実現したが、この場合、通信パケットのdataフィールドを拡張する形でseq、chkの２つのフィールド（図２参照）を追加した。

4．2 ファイル転送

ファイル転送は、利用者がワークステーション上に生成される専用のウィンドウを介して対話を行いながら処理が進められる。このとき、ワークステーション、ホスト・コンピュータの双方でファイル転送用のタスクが起動される。ワークステーション側のタスクは利用者からのコマンド入力および利用者へのメッセージ出力を行うとともに、ファイルの読出し／書込み、パケット処理を行う。ホスト・コンピュータ側のタスクはワークステーションより送られてくるパケット形式の指示に従って動作し、kermitのサーバ・モードに相当する役割を果たす。これにより、利用者はホスト・コンピュータ上のタスクの存在を意識せずにファイル転送を行うことができる。

4．3 ファイル転送多重化の効果

ＷＩＰの通信多重化方式は上記のアプリケーションによらず適用することができるが、以下では特にファイル転送の多重化について説明する。

ファイル転送の多重化が重要なのは、通常の端末回線ではファイル転送が一般に長時間回線を使用し、従来の端末インタフェース・プログラムではその間回線が占有され、転送が終了するまでホスト・コンピュータにアクセスできないという制限が生じるからである。これに対してＷＩＰでは、ファイル転送中もホスト・コンピュータと対話したり、ファイル転送自身を並行して処理することを可能にし（ファイル転送は送信／受信両方向の並行処理が可能である）、ホスト・コンピュータとの対話性を向上させている。

また、通信路を有効に利用する上でもこうした転送の多重化は有効である。kermitのようにハンドシェイクによってデータを転送する方式を採れば、回線はプロセッサがパケット処理を行っている間空きが生じることになるので、他のファイル転送を行うことによりその空きを埋めることができる。表３に、多重度によって転送効率がいかに向上するかを9,600 bpsのRS232C回線（全二重）を用いて測定した結果を示す。多重度３までは、ほぼ多重度に比例して転送効率が上がっている。また両方向の転送では、通信路の有効利用およびパケット処理の負荷の分散により転送効率が単方向の場合に比べ向上している。

多重度	転送効率（bytes/sec）	比
1	87.8	1
2	157.7	1.80
2*	265.7	3.03
3	263.8	3.00
4	330.4	3.76

*双方向

表３．ファイル転送多重化の効果

```
while (TRUE)
{
    if (pending input in line)
    {
        wid = receive_packet();
        write_task_input(search_task(wid));
    }
    for (i = each task)
        if (pending output in task[i]_output)
        {
            read_task_output(i);
            send_packet(task[i]_wid);
        }
}
```

図４．リモートＷＩＰのメッセージ切換えアルゴリズム

5．インプリメンテーション

5．1 リモートＷＩＰ

図４はリモートＷＩＰのメッセージ切換え部分（以後mpxと呼ぶ）をＣプログラム風に記述したものである。

mpxのインプリメントにまず必要となるのは、マルチタスキングができ、タスク間で通信する際、複数のタスクからのデータ入力が切り換えられることである。ここでは回線および各タスクからの入力データの切換えを non-blocking入力機構を使い、ポーリングを行って実現している。この方法によると、メッセージの切換えは単一のタスクで実行することができる。Ultrixにおけるインプリメンテーションでは、non-blocking入力機構としてシステム・コールselectを用いた。

このデータ入力の切換えを non-blocking入力を用いずにインプリメントするには一般に次のような方法が考えられる。

(1) 入力データの到着を割込みで通知する。この方法は、割込みが来ない限り入力タスクがプロセッサを占有しなくてもよいという利点がある。ただし、どのタスクからの割込みか識別することが必要である。

(2) それぞれの入力先に対応させて入力専用のタスクを起動させる。各タスクは入力先を示す情報を付加し、mpxに入力データを与える。この場合、n 個の仮想端末を得るのに合計 $2n+2$ のタスクがホスト・コンピュータ上で稼動することになる。

タスク間の通信はUnixのptyを用いた。ptyでは、従来の pipeと異なり、相手タスクに通常の端末と全く同じようにデータを与えることができるので、例えばvi、moreのような端末依存のプログラムの実行を容易にサポートすることができる。

利用者がＷＩＰを介してホスト・コンピュータと対話を行うには、さらに利用者レベルでコマンド・インタプリタを複数起動し、かつその入力先を端末以外に向け直す機能が必要である。こうした機能は Unixには存在するが、大型計算機のＯＳ（MVSなど）では制限が強すぎ、ＷＩＰの実現には不向きである。

５．２ ローカルＷＩＰ

ローカルＷＩＰのメッセージ切換え部分（ホスト・コンピュータの場合と同様mpxと呼ぶ）をＣプログラム風に記述すると図５のようになる。図５－(1)は利用者がホスト・コンピュータと対話する場合、通常入力が一時に１つの仮想端末に対してのみ行われることを利用し、これを集中的に処理する方式である。この方式はまた、端末からの入力がウィンドウではなく、キーボードという形でしか指定できないシステムでは必然的に用いられる。図５－(2)は、仮想端末の入力をすべて調べる方式である。ファイル転送を行う場合、複数の入力を時分割で処理するとすれば、この方式が自然に用いられる。

```
while (TRUE)
{
    if (pending output in line)
    {
        wid = receive_packet();
        write_window(wid);
    }
    if (pending input in tty)
    {
        wid = current_input_window;
        read_tty();
        send_packet(wid);
    }
}
```

(1) １つのウィンドウからの入力を調べる方式

```
while (TRUE)
{
    if (pending input in line)
    {
        wid = receive_packet();
        write_window(wid);
    }
    for (wid = each window)
        if (pending output in window[wid])
        {
            read_window(wid);
            send_packet(wid);
        }
}
```

(2) すべてのウィンドウからの入力を調べる方式

図５．ローカルＷＩＰのメッセージ切換えアルゴリズム

ローカルＷＩＰの実現に既存のＯＳが備えていなければならない機能をまとめると、(1) 利用者レベルで複数の仮想端末が利用でき、それらをウィンドウとして表示できること、(2) 回線と複数の仮想端末からの入力を切り換えることができること、が挙げられる。その外、ワークステーション上で各ウィンドウに直接対応させてタスクを動かす場合はリモートＷＩＰの場合と同様の条件が追加される。

５．２．１ Concurrent CP/M（CCP/M）上のＷＩＰ

CCP/Mでは、４つの仮想画面上で最初からコマンド・インタプリタが稼動している。我々は利用者と直接入出力を行うプログラムをこれらの仮想画面ごとに実行させ、それらのプログラムとmpxプログラムとが通信して、複数のタスクでローカルＷＩＰを構成する手法を採った。これにより、n 個の仮想画面をウィンドウ表示し端末として用いるとき全体で $n+1$ 個のタスクが起動されることになる。タスク間の通信はCCP

/Mが提供するキューを用いている。キューの入出力については、 non-blocking入出力機構が提供されていたのでこれを用いた。キーボードからの入力およびRS232C回線からの入力についてもBDOSあるいはBIOSが提供する専用のnon-blocking入力機構を利用している。

5.2.2 HI-UX上のＷＩＰ

HI-UXはUnix System Vと互換性をもつUnixシステムであるが、System Vにないselectシステム・コールをサポートしており、Hi-UX上のmpxはリモートＷＩＰと同様、selectを用いて単一タスクでポーリングを行う方式を採り実現した。従ってリモートＷＩＰのmpxと類似の構成になっている。select をもたない純粋のSystem Vの場合は、 5.1で述べた別の手法（各入力先に対応させてタスクを起動する）により実現する必要がある。入力行編集およびファイル転送のタスクとの通信はptyが提供されていないため pipe を用いた。ウィンドウに対する入出力は通常のファイルと同様に指定でき、この部分に関するプログラミングは比較的容易に行えた。しかしながら、使用可能なファイル・ディスクリプタ数の制限から、生成できるウィンドウの数はHI-UXの場合５に限られている。

5.2.3 Interlisp-D上のＷＩＰ

Interlisp-Dはキーボードからの入力を検出する関数READPをもち、またRS232C回線に対する読出し関数は、文字または文字列が読出し可能であればその値を、読出し可能でなければNILを返す仕様になっており、 non-blocking入力が可能となっている。Interlisp-Dでは mpxをこのnon-blocking入力機構を用いた、単一のタスクによるポーリングで実現している。

表１から分かるようにInterlisp-Dで実現したＷＩＰはＣ言語の場合に比べプログラムのサイズがかなり小さく、また実際のインプリメンテーションもわずか１日（通常の端末インタフェース・プログラムを実現した後）を要しただけであった。これは各種データの管理をリストを使って簡潔に表現できたことやウィンドウの取扱いが比較的上位レベルの関数を使って簡単に記述できたことによるものである。ただし問題点としては、Maclispが備えているtyi，tyoのような直接文字単位の入出力を行う関数がなく効率を低下させている点、キーボード入力について対応するウィンドウを特定できないためＷＩＰプログラム自身がこれを管理しなければならない点が挙げられる。

6．利用者インタフェース

ＷＩＰはまずワークステーション側で起動される。この時点でＷＩＰは単なるホスト・コンピュータの端末インタフェース・プログラムとして動作し、利用者はホスト・コンピュータにログイン可能となる。ログイン後ESC-nを入力すると、ホスト・コンピュータ側のＷＩＰが起動され、両者の間でパケットによるメッセージ交換が開始される。表４は利用者が指定できる主なコマンドの一覧を示したものである。システムによりサポートしていないものも存在するが、サポートしているものに関してはインプリメンテーションによらず、すべて表４の指定法に従う。ウィンドウの位置、大きさ、色などの属性の初期値は、（利用者プログラム・レベルでの）設定が許されていればＷＩＰが自動的に設定する。ただし、その後の変更はすべて既存のＯＳが提供する手段で行う。これにより、システムの作成が容易になるとともに、利用者も新たに操作法を修得する必要がない。

コマンド	機能
ESC-n	Create a new window.
ESC-k	Kill the current window.
ESC-q	Quit. End the WIP session.
ESC-l	Enter/Exit line editing mode.
ESC-f	Create a new window and execute file tranfer.
ESC-h	Display the help message.

表４．主なＷＩＰコマンド

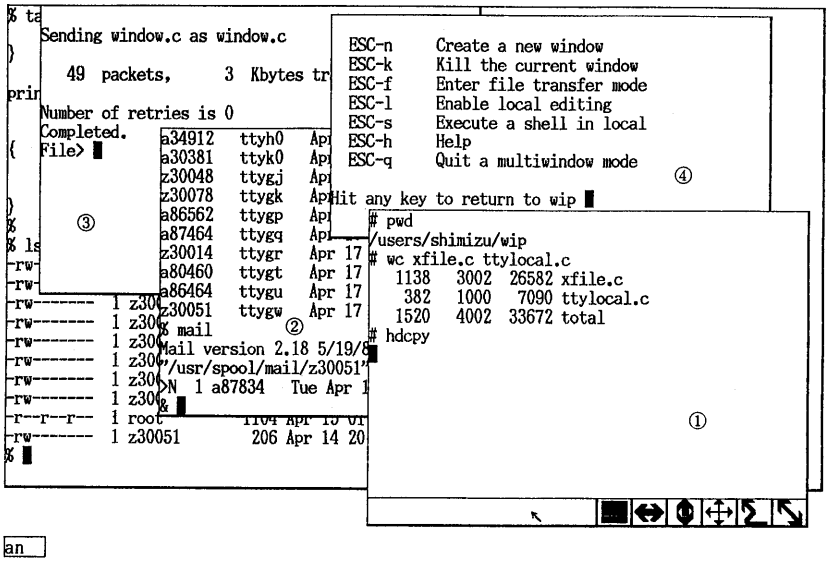


図６．Hitachi 2050上でのＷＩＰの画面のスナップショット

ＷＩＰの実際の使用例として、図６にHitachi 2050 上での画面のスナップ・ショットを示す。最後方のウィンドウはＷＩＰ立ちあげ時のウィンドウで、システム全体に関係するメッセージはここに表示される。他のウィンドウはＷＩＰにより生成されたウィンドウで、2050上のshellが稼動中のもの（①）、ホスト・コンピュータと交信中のもの（２，３）およびヘルプ専用ウィンドウ（４）がある。プロンプトFile>が表示されているウィンドウ（３）はファイル転送のためのウィンドウで、ワークステーション側でファイル転送を専門に行うタスクが稼動している。転送中、画面には送られた byte数、パケット送出の再試行回数等が表示される。実際のウィンドウは機能別に色分けされている。

７．あとがき

ワークステーションが備える既存のマルチウィンドウ機能を利用し、ホスト・コンピュータのマルチウィンドウ端末を実現するシステムＷＩＰを提案し、実際にホスト・コンピュータと３つのワークステーション上でインプリメントを行った。ＷＩＰではＯＳ、言語の異なるワークステーションに対し、それぞれ別のプログラムが必要となるが、通信プロトコル・レベルで移植性をもたせており、また UnixのようにＣ言語とともに広く用いられているシステムでは他機種への移植も容易と考えられる。我々は今回、ホスト・コンピュータ、ワークステーション間の接続にRS232C回線を用い、その上に直接通信部分をインプリメントする方式を採ったが、安価でしかも広く用いられているRS232C回線を利用することにより手軽に本システムを移植あるいはインプリメントすることが可能になっている。今後、実際のネットワークを使ったこの種の機能の実現について考えていきたい。

我々はまた、ＷＩＰの機能を Lispシステムに組み込んだ新しい Lispのプログラミング環境の構築に関し研究を進めている。

参考文献

1. 日立クリエイティブ・ワークステーション２０５０ システム概説，マニュアル2050-1-001，日立製作所 (1986).
2. Pike, P. : The Blit : A Multiplexed Graphic Terminal, AT&T Bell Lab. Tech. J. Vol.63, No.8, Part 2, pp.1607-1632 (1984).
3. Cruz, F. D. and Catchings, B. : Kermit : A File-Transfer Protocol for Universities, BYTE Vol.9, No.6, pp. 255-278, and No.7, pp.143-403 (1984).