ホスト計算機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†††

本稿では，ワークステーション上でホスト計算機のマルチウィンドウ端末を実現し，ホスト計算機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 (Window Interface Program) について述べる．WIP は，ホスト計算機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 は無手順の通信回線による従来の端末インタフェース・プログラム (Terminal Interface Program) と同様の簡便さを有したユーザ・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5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や望ましい機能について，実際に実現した経験を交えながら考察を行う．

1. まえがき

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備えたマルチウィンドウ機能は，優れたマン・マシン・インタフェースを実現し，既に 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提供する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の一つとして，ホスト計算機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，単にホスト計算機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，ホスト計算機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，ホスト計算機のマルチタスク処理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，ウィンドウ管理，ウィンドウ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計算機の負荷を削減することができると同時にホスト計算機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，既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），ホスト計算機に対する一つのセッションの中でマルチタスク処理を行う．したが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]や MIT の X-window システム[3]があるが，本研究では別なアプローチとして，

（1） ワークステーション上の既存の異なる OS にユーザ・レベルのプログラムとして容易に実現できる簡便なシステムの開発を目的とし，実際に表1に示す5機種のワークステーション上で実現を行ってその容易性を実証した．

（2） 無手順の通信回線で，ホスト計算機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．これは，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[4] の機能拡張の一つの方向を示すものである．

（3） ワークステーション，ホスト計算機双方において，アプリケーションに応じたタスクを自由に WIP に組み込んで動かせるようインタフェースを整

† Implementation of Multiwindow Terminals for Supporting Multitasking on Host Computers by KENTARO SHIMIZU (Department of Information Science, Faculty of Science, University of Tokyo) and HARUHISA ISHIDA (Computer Centre, University of Tokyo).

†† 東京大学理学部情報科学科
††† 東京大学大型計算機センター

表 1 実現したシステム
Table 1 Implementations.

マシン	OS	記述言語	プログラム・サイズ
NEC PC 9801	Concurrent CP/M	C, 8086 アセンブリ言語	1,030 行
HITACHI 2050	HI-UX	C	950 行*
HITAC E 7300	UNIRIS/UX	C	1,110 行*
OKI if 1000	UniPlus+ SYSTEM V	C	1,150 行*
Xerox 1100 SIP	Interlisp-D	Interlisp-D	570 行
Vax-8600	Ultrix-32	C	800 行

* は拡張機能をもつがその部分は除く

え拡張性を持たせた.

(4) この種のシステムをユーザ・レベルで実現する上で既存の 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, 実際に実現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 の実現について比較的詳細な議論を行う. また, 6章では WIP のユーザ・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 によるホスト計算機とワークステーションとの結合を示したものである. ホスト計算機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1対1に対応し, ユーザはワークステーション上のウィンドウ i を介してホスト計算機のタスク i と交信する. すなわち, ホスト計算機のタスク i に対応して仮想端末 i が生成され, そ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. ホスト計算機とワークステーションの間は物理的に1本の通信回線で結ばれているだけでよく, 通信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, すなわちホスト計算機上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 (ウィンドウ番号識別フィールド) をもつ.

WIP はホスト計算機, ワークステーションの双方で稼動する. 以下, ホスト計算機側の 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, ウィンドウ番号識別フィールドを調べ, 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番号をウィンドウ番号識別フィールドに入れ, パケット化してワークステーション側に送出する. 一方, ローカル WIP はリモート WIP とほぼ対称的な構成で, ホスト計算機より送られてきたパケットを切り分け, 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計算機に送出する.

3. メッセージの転送方式

3.1 パケットの形式

WIP では, 図2に示すようなパケット形式でメッ

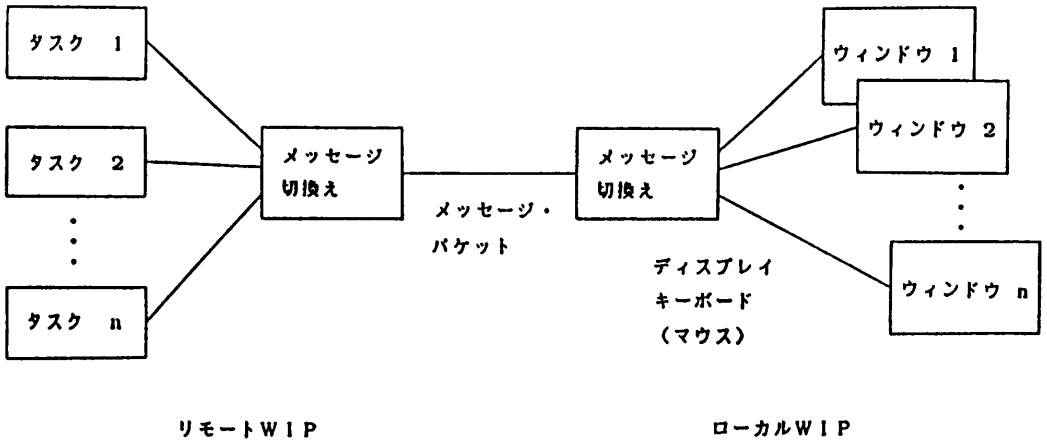
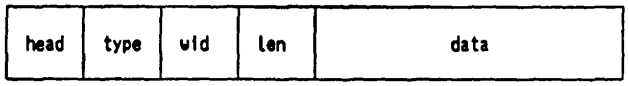
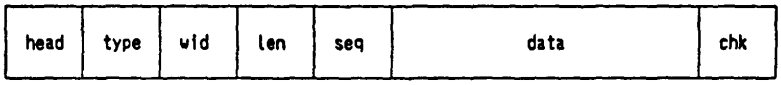


図 1 WIP システムの概略
Fig. 1 Outline of the WIP system.

(I) 通信パケット



(II) ファイル転送パケット



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

図 2 メッセージ・パケットの形式
Fig. 2 Message packet formats.

セージのやりとりを行う．すべてのパケットは，回線多重化のためのウィンドウ番号識別フィールドを持つ．パケットには，通常の対話に用いられる通信パケットとファイル転送に用いられるファイル転送パケットの2種類がある．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[4] のパケット形式にウィンドウ番号識別フィールドを追加した形でもある（他のフィールドの意味とパケット・タイプは kermit と同じである）．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，また data フィールド中の非印字可能文字は，2バイト化し印字可能文字に変換して転送する．これは，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．

表 2 通信パケットのタイプ
Table 2 Types of the communication packet.

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	A	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

ローカル WIP からリモート WIP に対し，表2に示す各種の要求が送られ，リモート WIP はこれらの要求に対し，ローカル WIP に ACK, NAK のいずれかを返す．しかし，通常のメッセージ・データについては，今のところ応答性を考慮して，ファイル転送のような ACK/NAK によるハンドシェイクやチェックサムの照合は行っておらず，head フィールドを用いて同期をとっているだけである．

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．この場合，パケット化の処理がネックとなるかパケットの転送自体がネックとなるかは両者の相対関係によって決まる．ここでは，特にプロトコルの性能に着目して転送する文字数オーバヘッドについて考えることにする．

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．

ワークステーション側で入力された文字データを1文字単位でホスト計算機に送りホスト計算機によるエコー・バックをそのまま利用するとき，1文字入力するごとに5バイトのパケットが両方向に転送されることになり，P は大幅に増大する．これを避けるには，ワークステーション側でローカル・エコーを行い，入力を行単位にすればよい（ホスト計算機はエコー・バックは行わない）．もちろん，ホスト計算機からワー

クステーションへのメッセージ出力があるので，効果は入力行の長さに比例するところまではいかず，実際の端末セッション（プログラムの編集・コンパイル・実行など）を 30 回以上にわたりモニタリングして測定した結果によると P は 1/3 程度 (0.11/0.33) になった．ただし，前者の場合パケット化やメッセージ切換えのための処理上のオーバヘッドが増大するので，実質的な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（例えば r は $1/l$ 以下，s は無視できる程度に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得る．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，信頼性を考慮し，上記のようなパケット転送方式を採用した．

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，仮想端末ごとに専用のタスクを走行させ，それらがメッセージ切換え部と通信する形で実現される．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一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．通常の対話セッションではコマンド・インタプリタのシェルがバックグラウンドで並行して実行される．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 の二つのフィールド（図 2 参照）を追加した．

4.2 ファイル転送

ファイル転送は，ユーザ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，ホスト計算機双方で稼動する専用のタスクの間で kermit のサーバ・モードに似たプロトコルで処理が行われる．すなわち，ワークステーション側のタスクはユーザからのコマンド入力およびユーザ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．ホスト計算機側のタスクはワー

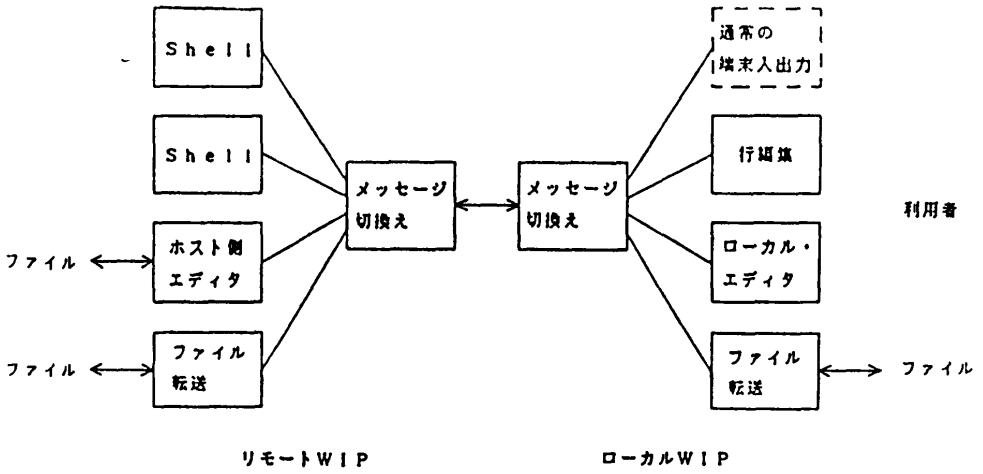


図 3 アプリケーションに応じたタスク間通信
Fig. 3 Intertask communication for several applications.

表 3 ファイル転送多重化の効果
Teble 3 Effect of the multiplexed file transfer.

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

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する．これにより，ユーザはホスト計算機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 では，上で述べた通信多重化方式により，ファイル転送中もホスト計算機と対話したり，ファイル転送自身を並行して処理することを可能にし（ファイル転送自身は送信／受信両方向の並行処理が可能である），ホスト計算機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，通信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 RS 232 C 回線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，通信路の有効利用ならびに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て向上している．

5. 実現方式

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 の実現にまず必要となるのは，マルチタスク処理が可能で，タスク間通信が効率よく行え，さらにその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，通信回線および各タスクからの入力データの切換えを非封鎖入力機構を使ったポーリングにより実現している．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．非封鎖入力機構を用いずに実現するには，一般に，①入力データの到着を割込みで通知する，②各入力先に対応させて入力専用のタスクを起動させる（この場合，n 個の仮想端末を得るのに合計 $2n+2$ のタスクがホスト計算機上で稼動することになる），といった方式が考えられる．

```
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
Fig. 4 A message exchange algorithm for the remote WIP.

我々は，リモート・システムとして，Ultrix (4.2 bsd Unix 上位互換) を搭載した Vax 8600 を選んだ (表 1 参照) が，Ultrix では，非封鎖入力機構として，システム・コール select を用いることができた．タスク間の通信には疑似 tty (pty) を用いた．pty では，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，vi，more のような端末依存のプログラムの実行をそのままサポートすることができる．

ユーザがホスト計算機と対話を行うには，さらにユーザ・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 や VMS には存在するが，汎用大型計算機の 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 (mpx) をC言語風に記述すると**図 5** のようになる．ローカル WIP では，仮想端末に対する入出力の切換えが行われる．図 5 から分かるようにその構成はリモート側とほぼ対称的である．したがって，リモート側 WIP と同様のメッセージ切換え方式が適用される．ただしこの場合，入力が一時に一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 (図 5-(1)) と，仮想端末の入力をすべて調べる方式 (図 5-(2)) が考えられる．後者の方式は，例えばファイル転送で複数の入力を時分割処理する場合に用いられる．

ローカル WIP を既存の OS に実現する際，その OS が備えていなければならない機能を挙げると，①ユーザ・レベルで複数の仮想端末が利用でき，それら

```
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
(1) Scanning the input from one window at a time

```
while(TRUE)
  { if(pending input in line)
     { wid=receive_packet();
       write_window(wid); }
    for(wid=each window)
       if(pending output inwindow[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
(2) Scanning the input from all the windows at a time

図 5 ローカル WIP のメッセージ切換えアルゴリズム
Fig. 5 A message exchange algorithm for the local WIP.

をウィンドウとして表示できること，②通信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，四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．我々はユーザ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 個の仮想画面をホスト計算機の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，非封鎖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 RS 232C 回線からの入力についても BDOS あるいは BIOS が提供する専用の非封鎖入出力機構を利用している．

5.2.2 UNIX 上の WIP

現在，ワークステーション上の WIP については，表1に示した System V と互換性のある三つの UNIX システム上で実現を行っている．このうち，HI-UX は Berkeley 版 UNIX と同じ機能をもった select をサポートしていたため，これを使って mpx を単一のタスクで実現した．他のシステムについては，5.1 節で述べたように，各入力先に対応させて入力専用のタスクを動かす方法を採っている．mpx と入力行編集ならびにファイル転送用のタスクとの通信には pipe を用いた．どのシステムでも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，この部分の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，例えば HI-UX の場合生成できるウィンドウ数は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 を持ち，また RS 232C 回線に対しても非封鎖入力が可能になっている．Interlisp-D では mpx をこれらの非封鎖入力機構を用いた，単一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 は他システムの場合に比べプログラムのサイズがかなり小さく，また実際のコーディングもわずか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構造を使って簡単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. ユーザ・インタフェース

WIP はまずワークステーション上で起動される．この時点で WIP は単なるホスト計算機の端末インタフェース・プログラムとして動作し，ユーザはホスト計算機にログイン可能となる．ログイン後 ESC-n を

表 4 主な WIP コマンド
Table 4 Principal WIP commands.

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.

入力するとホスト計算機側の 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 はユーザ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実現によらず，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，(ユーザ・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，ユーザも新たに操作法を習得する必要がない．

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 立ち上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 上のシェルが稼動中のもの (①)，ホスト計算機と交信中のもの (②，③)，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．画面には，転送されたバイト数，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

WIP のような機能があれば，ワークステーション上の処理とホスト計算機上の処理を並行して実行できるだけでなく，ワークステーション上においてホスト計算機を使った複数の仕事を同時に進行させることが可能になるので，計算機の利用効率がかなり上げられるものと思われる．例えば，ワークステーション上で編集したプログラムをホスト計算機に送ってコンパイル・実行するという一連の作業を複数のモジュールに対してパイプライン的に並行して行うことが可能である．また，分散エディタのようなホスト計算機とワークステーションが協調して働く，両者のプロセス間通信に基づくアプリケーションをそのまま多重化して利用することができる．現在，我々は 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

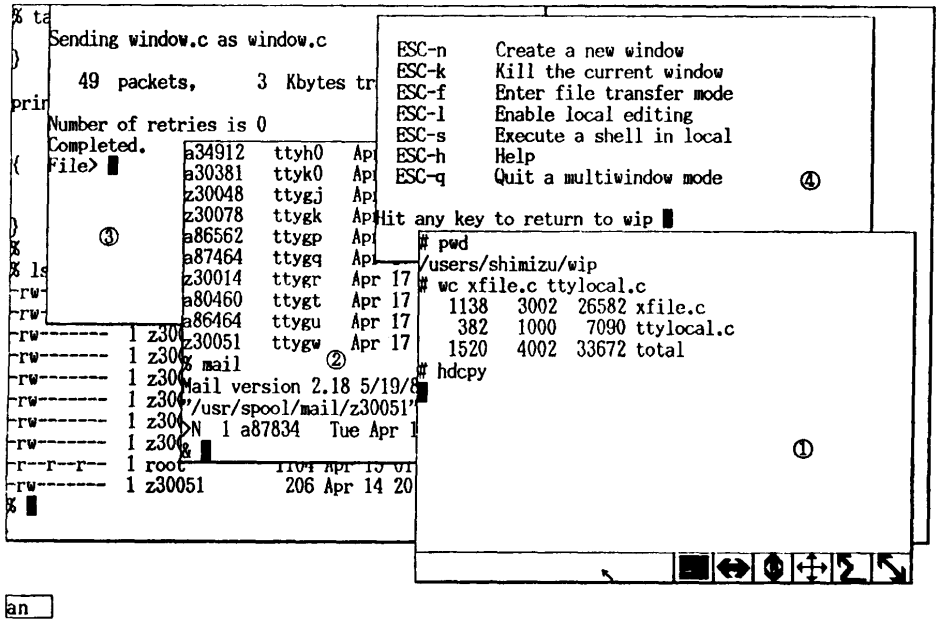


図 6 Hitachi 2050 上での WIP 画面のスナップショット
Fig. 6 Snapshot of the WIP screen on Hitachi 2050.

ング環境の構築に関し研究を進めている．

WIP では，OS，言語の異なるワークステーションに対し，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は容易である．実際，Hitachi 2050 から HITAC E 7300，OKI if 1000への移植ではウィンドウ関係の100行程度のコード変更で済んでいる．ネットワークや特殊なハードウェアを用いずに，しかも比較的簡単なソフトウェアで，上に述べたような作業環境を提供できることが WIP の重要な特徴である．今後は，他機種への移植を進めるとともにネットワーク・システムへの適用や複数マシンとの接続について考えていきたい．

謝辞 WIP の HITAC E 7300 および OKI if 1000 への移植を中心となって行った日立製作所の杉山武信氏に感謝します．また査読委員の適切なコメントに対し謝意を表します．

参 考 文 献

1) Cruz, F. da: *Kermit, A File Transfer Protocol*, Digital Press (1986).
2) 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).
3) Pike, P.: The Blit: A Multiplexed Graphic Terminal, *AT & T Bell Lab. Tech. J.*, Vol. 63, No. 8, Part 2, pp. 1607-1632 (1984).
4) Scheifler, R. W. and Gettys, J.: The X Window System, *ACM Trans. on Graphics*, Vol. 5, No. 2, pp. 79-109 (1986).

(昭和 61 年 12 月 15 日受付)
(昭和 62 年 7 月 9 日採録)

清水謙多郎（正会員）

昭和 32 年生．昭和 55 年東京大学理学部情報科学科卒業．昭和 60 年同大学大学院博士課程修了．理学博士．同大学大型計算機センター助手を経て，理学部情報科学科助手となり，現在に至る．LISP マシンの開発，ジョセフソン計算機向きアーキテクチャの設計に従事．現在は，分散オペレーティング・システムの研究に関与．日本ソフトウェア科学会会員．

石田 晴久（正会員）

昭和 11 年生．昭和 34 年東京大学理学部物理学科卒業．昭和 36 年同修士課程修了後，フルブライト留学生として渡米．昭和 39 年アイオワ州立大より PhD．その後 MIT 研究員．昭和 41 年電気通信大学助教授．昭和 45 年東京大学大型計算機センター助教授．昭和 57 年同教授．ACM，IEEE などの会員．最近は分散処理や卓上出版に関して研究開発を行っている．