高機能ワークステーションのＯＳについて
—単一ユーザ多重環境システム—

慶應義塾大学理工学部
斎藤信男　村井　純　中村　修　水場英世
ソフトウェアリサーチアソシエイツ
岸田孝一　太田靖彦　栗原正利

１．はじめに

最近の半導体技術の進展やシステム統合技術の発展により，いわゆるスーパーパーソナルコンピュータあるいは高機能ワークステーションが多く見られるようになってきた。　この最初のシステムは，Ｘerox ＰＡＲＣのAlto マシン〔1〕であり，その延長上にあるDorado，Dolphine，Ｓtar 8000などが開発されている。又，最初の商業ベースの高機能ワークステーションであるＰＥＲＱ，Apollo Domain，ＳＵＮ，Ｌisa などは，汎用のものであり，Ｓymbolics社のＬisp マシンは，専用言語のものと分類できる。

このような高機能ワークステーションは，その特徴を生かして，いろいろの分野で使われる可能性がある〔2〕。　たとえば，ソフトウェア開発環境，ＣＡＤ／ＣＡＭ，ＣＡＩ，ビジネスグラフィクスなどのオフィスオートメーションなど，その用途は広がりそうである。このような，高機能ワークステーションの一般的な構成を，図１に示す。　従来のパーソナルコンピュータと違うところは，高機能，高解像度のビットマップディスプレイとポインティングデバイス，大容量のファイル記憶装置，高機能のネットワークインターフェースなどである。図形情報の会話的処理が高性能で行なえることから，ユーザインターフェースの改善が大巾に進み，より使い易い情報システムのワークステーションとして役割を果たすことが期待される。

このような高機能ワークステーションのソフトウェアは，現在多くのメーカーで開発されているが，必ずしもユーザの各種用途に対して満足のいくようなものではない。前述の特徴を十分生かしたソフトウェアを実現するためには，その基礎となるＯＳの開発が何と言っても重要である。　ＳＵＮワークステーションは，Unix を装備し，それを発展させてゆく方針であろうが，新しい器の高機能ワークステーションには，新しい考えに基づくＯＳがふさわしい。

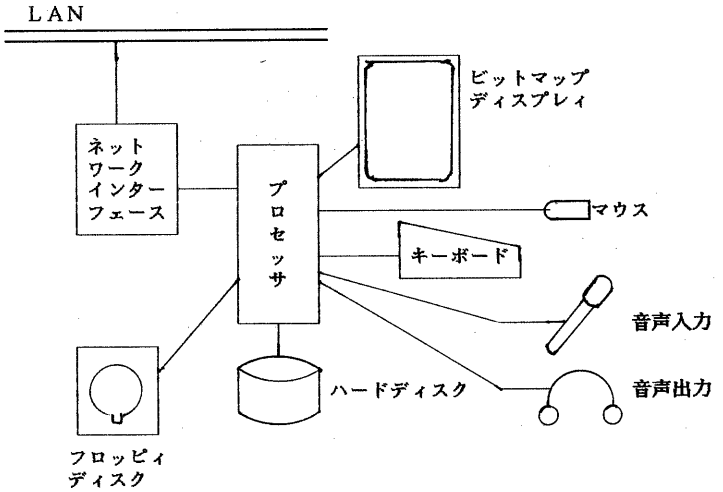

図１　高機能ワークステーションの構成例

ここでは，主としてソフトウェア開発環境で使用することを前提として，高機能ワークステーションにふさわしい新しいＯＳの設計方針について見解を述べ，その問題点などを整理する。なお，この方針は，他の用途に使うワークステーションに対しても，充分その基礎として働くものと考える。

2．単一ユーザ多重環境のオペレーティングシステム

2．1．何故単一ユーザ多重環境のＯＳか

高機能ワークステーションは，ユーザとのインターフェースの向上を目的としているので，多重ユーザよりも単一ユーザの環境で使うことが望ましい。　勿論，ネットワークを介した分散処理系が考えられ，この場合，システム全体としては多重ユーザのシステムになる。　ロードバランスを実施すれば，必然的に複数のユーザのプロセスが１つのワークステーションで実行される。　しかし，単一のワークステーションに対する基本方針として，単一ユーザのＯＳを考える。

１つのプログラムの実行に際して，それが使用するファイル，入出力機器などの計算機資源が使われるが，これらの資源の集合をその環境（実行環境）という。　環境は計算機資源の名前，型，それらへのアクセスパス，アクセス権などにより定義される。　この環境は，１つ１つのプログラムにそれぞれ異なったものを与えてよい。　したがって，一人のユーザが１つのワークステーションに於いて複数の仕事を同時に実行するとき，複数個の環境が同時に存在することになる。　以上の考察により，高機能ワークステーションのＯＳは，単一ユーザ多重環境のＯＳであるといえる（図２）。

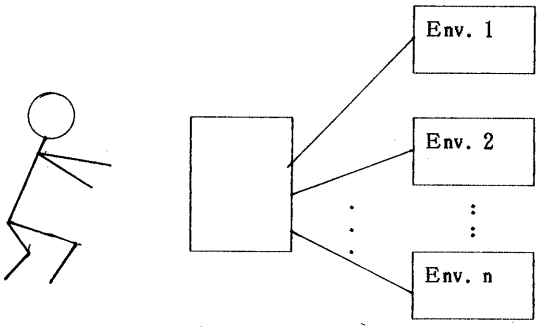

図２　単一ユーザ多重環境のＯＳ

単一ユーザ多重環境のＯＳに於いては，次の３つの問題を考えねばならない。

（1）　環境の定義とその管理
（2）　バージョン管理
（3）　ユーザインタフェース

2．2　環境とその管理

ＯＳにおいて，１つのプログラムの実行ごとに環境を設定し，それを介して資源へのアクセスなどの管理を行うことの利点は，次の通りである。

（1）　プログラムのモジュラリティが向上する。

プログラムの実行時に使う計算機資源を一緒にして環境として定義してあるので，従来以上にモジュラリティが向上する。

（2）　計算機資源に対するアクセス制御が厳密かつ柔軟におこなえる。

ネットワークを介して遠隔地の資源をアクセスする場合，事前にアクセス権のチェックが出来れば，能率が良い。　環境の中に，資源名やアクセスの要求等を指定しておけば，そのようなチェックが可能である。

（3）　ロードバランシングの単位となる。

ネットワークを介してロードバランスを行なう場合，実行環境も含めたプログラムを他のサイトに割り当てることにより，確実にその実行が保証される。　又，サイトの違いを環境というインタフェースにより吸収できる。

ユーザは，各プログラムやソフトウェアパッケージに対して，その環境を定義し，又，既にある環境の再利用が可能となるような機能を使いたい。　又，プログラムの実行時の任意の時点で，環境を固定し，その保存と再利用が柔軟におこなえる機能が必要である。

2．3．バージョン管理

環境は，計算機資源の集合であり，ファイルだけでなく，プロセス，端末，ユーザ等々がその要素になる。

一般に，計算機資源は状態をもち，その状態が時間とともに変わってゆく。　ここで，ファイルや記憶の内容なども状態と見なす。　バージョンとは，この状態遷移の時間的経過を適当な間隔でとらえたものと考える。

ユーザが必要に応じてバージョンを指定し，それを格納したり検索したりするためには，システムが統一的なバージョン管理をしなければならない。　バージョン管理の問題点は，次の通りである。

（1）　バージョン設定のタイミング

バージョンの設定の間隔があまり小さいと，前後のバージョンとの差の認識があいまいになり，また，バージョンの数もふえてしまう。　このタイミングの設定を，ユーザの操作の上位のレベルに合わせて自動的に行なうと便利である。

（2）　バージョンの名前付け

バージョンの格納と，その検索との操作において，それらを識別するための名前付けは，柔軟に行えなくてはならない。

（3）　バージョンの整理

バージョンは，１つの計算機資源に対して時間的に増大してゆく。　したがって，システム又はユーザの環境の中で意味の無くなったバージョン（たとえば，しばらく参照はされないもの）については，ごみ集めのような機能を働かせて整理することが必要である。

2．4．ユーザインタフェース

オペレーティングシステムも，ユーザから見れば１つのソフトウェアシステムであり，その使い心地などを決めるユーザインタフェースは重要な問題である。

Unix では，コマンドインタープリタ（shell ）により，柔軟なユーザインタフェースの実現を目指している。

高機能ワークステーションの１つの目的は，ビットマップディスプレイを利用した巧妙なユーザインターフェースの実現であり，多重環境のオペレーティングシステムでも当然その大幅な利用が１つの目的となる。

我々が目指しているＯＳのユーザインターフェースの問題は，次の通りである。

（1）　多重環境とウィンドウ制御

ビットマップディスプレイを使用すると，多重ウィンドウの表示が可能となる。多重環境のシステムでは，環境ごとにウィンドウを設定し，その表示を何らかの方法で行なう。ユーザは，そのウィンドウによって実行経過の監視や環境との入出力を行なう。

（2）　ウィンドウと編集系

ユーザがシステムと会話をするときの最前線がユーザインターフェースであり，そこでは文字列や図形等の情報のやり取りが行なわれる。このとき，それらの情報の編集も，このユーザインターフェースで自由に行なえると便利である。この編集は，Ascii文字，漢字，画像など，その対象に応じて自由に切り替えられることも必要である。また，異なったウィンドウ間での情報のやり取りも，この編集系を介して行なえると都合が良い。

3．環境の管理

オペレーティングシステムにおいて，ユーザが1つの仕事をする時に，その作業環境を明確に定義して，その保存や回復をユーザの指定に応じて適宜行なうことは，多重アクセスのオペレーティングシステムや，ネットワークを介した分散処理システムにおいて不可欠のものである。　環境の管理は，次のような利点をもつ。

（1）　コマンドの実行に対する柔軟なユーザインタフェースを提供する。

（2）　共有資源に対するアクセス制御の厳密化が可能となる。

（3）　資源に対する名前付けが自由に行なえる。

（4）　ネットワークにおいてロードバランスが効率よくできる。

（5）　ネットワークにおいて並列処理が効率よく行なえる。

（6）　ソフトウェアのモジュール化が向上する。

3．1．　環境の定義

環境とは，計算に於いて使用する計算機資源の集合である。　資源には，識別名と型が与えられる。1つの環境の中では，識別名はその中だけ適用する局所的名前であり，他の環境での参照は，import／export機構によって行なう。　環境の宣言の例を，図3に示す。

```
environment env
{
    object
    {
        file            x,y,z;
        device          tty,lp;
        user_interface  myshell;
        process_group   procl;
        command         mycc;
            .
            .
            .
    }
    initialization
    {
        < myenv_init;
        export  ...  to    his_env;
        import  ...  from  her_env;
    }
}
```

図3　環境の宣言

環境に宣言されている資源は，ファイルと同様に構造をもっていてもよい。　1つの環境は，それが必要になった時，具体化　(instantiation) される。　そこでは，資源の生成，複写，アクセスパスの設定など必要な操作が行なわれる。

3．2　環境の管理

環境の各具体値には，図4に示すように，それぞれ自分自身の環境制御系が存在し，必要なコマンドの処理などを行なう。　これは，通常のコマンドインタープリタ（shell ）に対応する。　環境制御系は，最初に必要な資源の生成，又は複写を行なう。　又，アクセスパスの設定においてはアクセス権についてのチェックを行なう。分散処理システムに於いては，作業開始前のこのチェックは特に有益である。　他の環境からの資源のimportは，そのアクセスのモードにより，資源自身の転送か，アクセスパスの設定を行なう。importコマンドは，環境の定義において指示するが，プログラムの実行時にオンラインで指示することもできる。

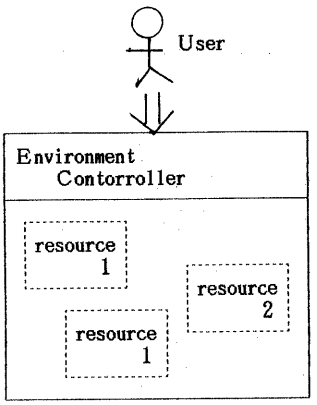

図4　環境制御系

環境の保存は，ユーザの指定によって行なう。　保存すべき資源のバージョンを識別し，回復時にそれを正しく得られるようにする。　資源に型によって，そのままでは保存できないもの（たとえばプロセス）があるが，その場合には状態の記述子を保存し，回復時に対応する資源をできるだけ再現するようにする。

4．バージョン管理

ソフトウェアの開発，管理，保守を行なう上で小型，大型に限らずＯＳにおけるバージョン管理機能の重要性と必要性が認識されるようになってきた〔4〕。　ここでは，高度なユーザインタフェースを提供し，会話的に使用する高機能ワークステーションのためのＯＳにおけるバージョン管理の特質について述べる。

4．1．バージョン管理の意義

計算機の上で行われる作業はいくつかの環境の組合せの上になりたっている。　ユーザは，過去の任意の時点の環境を再現したくなることがある。　例えば，新しいバージョンの言語処理系が提供されたとき，その言語処理系に依存した応用プログラムはもはや実行できなくなる可能性がある。このような場合，ＯＳにバージョンを管理する機能があれば，容易にもとの環境を再現し過去の応用プログラムを実行することができることがある。　このように，バージョン管理は環境の管理と密接に関連し，ソフトウェアの移植性を高めし，ソフトウェアの生産性を向上させることができる。

4．2．バージョン管理の対象

従来のＯＳにおけるバージョン管理の対象といえば，ファイル程度しか考えられなかった。　特にソースコードレベルでのバージョン管理〔5〕を考えることが多かった。　これは，ソースコードさえあればオブジェクト・モジュールはいつでも作成可能であるという発想を基にしている。　いずれにしても，ファイルのみしかバージョン管理の対象としないという点では，これからの高機能ワークステーション用のＯＳの機能としては不十分であり，自ずとその対象を広げて考えざるおえなくなる。　高機能ワークステーションのためのＯＳが行うバージョン管理の対象としては，次のようなものをあげることができる。

ファイル
プロセス
端末
ウィンドウ
環境
(オペレーティングシステム)
・・・・・・・

ここであげたたように，バージョン管理の対象となるものは，比較的静的でその対象を扱う際節目のようなものが存在するものとなる。　もともとバージョン管理は環境の管理を実現することが一つの目的であったが，環境自身も再帰的にバージョン管理を行なうことも考えられる。

4．3．バージョン管理の方法

バージョン管理機能を実現する上で，いくつか解決すべき点あるいは選択しなければならない点がある。

(1)　バージョンの登録

バージョンの登録を誰が，いつするかということは一つの大きな問題点となる。　例えば，ユーザが新しいバージョンを指定して登録するのか，あるいはＯＳが自動的に行うのか，またＯＳが自動的に行なう場合，どのような要因または動機をもとに新しいバージョンを登録するということである。現在，いくつかのメインフレームおよびスーパーミニコンのＯＳではファイルに関するバージョン管理の機能を取り入れており，それらは編集系などでユーザがファイルの内容を更新する度に新しいバージョンを登録する。

(2)　バージョンの識別

過去に登録したバージョンをＯＳが自動的に操作するだけならば，ＯＳの内部で識別できる簡単な識別機構でよいが，ユーザがある対象の任意のバージョンを抽出するためには，識別機構についても考察する必要がある。　まず，バージョンの履歴をもった１つの対象は１つにまとめて認識できねばならない。　次に，各々のバージョンの識別する方法であるが，これは，バージョンをそれを登録した時点の環境とともに識別できるようにすることが実現法の一つとしてある。　しかし，このような機構を実際のＯＳにどのように実現するかは，なお一層の検討を必要とする。

(3)　バージョンの保管と整理

上の２つの問題は比較的論理的であったが，今度は登録されたバージョンを実際にどのように保管するかという問題について述べる。　各バージョンは最終的にはファイルのようなものとして２次記憶装置に保存されることとなる。　各バージョンを全くそのままの形で保存することは，空間的に効率が悪い。　したがって，バージョン間の差分をとることが空間の節約になる。　また，必要のないバージョン，すなわち，システムに於いてどこからも参照されなくなったバージョンについては，自動的に回収して削除あるいは，バックアップファイルへ退避させるなどする。　計算機の資源には，論理的構造が存在する。　これを利用すると，Unix のmakeの様にバージョンの自動生成，削除などができる。

5．ユーザインタフェース

ワークステーションでのユーザインタフェースを考える場合，ハードウェア機器としては，ビットマップディスプレイのような高機能なディスプレイや，マウスなどのポインティングデバイスを利用することは，ワークステーションの性格や設計方針などから考えて，不合理な前提ではないであろう。

実際，これら高機能な装置を利用した多重ウィンドウによるユーザインタフェースは，近年，多くのシステムで実現されている。　また，ＣＲＴを利用した多重ウィンドウが使える編集系なども，使いやすいものとして多くの場所で利用されている〔6〕。

ワークステーションでのユーザインタフェースも，この多重ウィンドウの概念を取り入れ，より使いやすいものを考えなければならない。　ここでは，ワークステーション用ＯＳでは，どの様なユーザインタフェースにすればよいかについて，多重ウィンドウの機能も含めながら考察する。

5．1．使いやすいユーザインタフェースとは

使いやすさといってもいろいろなことが考えられるが，一言でいってしまえば，“やりたいことが簡単に行なえて，自分の仕事に対してのフィードバックを得やすいこと”，といえる。　たとえば，テキストの編集を考えてみると，画面編集系は，自分のやった事に対する結果，すなわち，テキストの修正結果をすぐに見ることができるので，行編集系にくらべて，使いやすい。　修正が間違っていれば，すぐに再度修正することが容易である。　すなわち，フィードバックを得やすいのである。　このことをもうすこし考えると，編集系という処理系に対して“修正する”，という入力をし，その結果である出力がすぐに見える。

このように考えると，使いやすさとは，以下の様な尺度でみることができる。

(1)　処理系の入力に対して出力が，どのくらい早く見ることができるか。

(2)　処理系への入出力に対しての，あつかいやすさはどうであるか。

Unix が使いやすいシステムであるという理由の１つとして，（2）の入出力のあつかいやすさをあげることができる。

Unix では，入出力はすべて，ファイルという形によってあつかわれている。キーボードもＣＲＴもプロセスからは，ファイルとしてあつかわれ，これらのファイルへの口のつげかえによって，Unix は使いやすいシステムになっている〔7〕〔8〕。

5．2．ワークステーションにおけるユーザインタフェース

ワークステーションでは，ユーザインタフェースの装置として，ビットマップディスプレイなどを利用し，多重ウィンドウによるインタフェースを提供することは前に述べた。　この多重ウィンドウの機能を使ったよりよいユーザインタフェースの１例として，以下の様なものを提案する。

（1）　ウィンドウとは，バッファの１部分を，ユーザに見せる部分である。

（2）　バッファとは，作業用ファイルである。　すなわち，プロセスのＩ／Ｏの対象となりえる場所である。

（3）　ウィンドウ内では，編集が容易にできる。　また，ウィンドウ間の情報（たとえば文字情報）のやり取りが容易に行なえる。

このようなユーザインターフェースの概略を図５に示す。　ウィンドウは，バッファの内容をビットマップディスプレイ上に表示したものであり，その表示の仕方は，重ね合わせ，拡大，縮小の操作によって指示されるものとする。

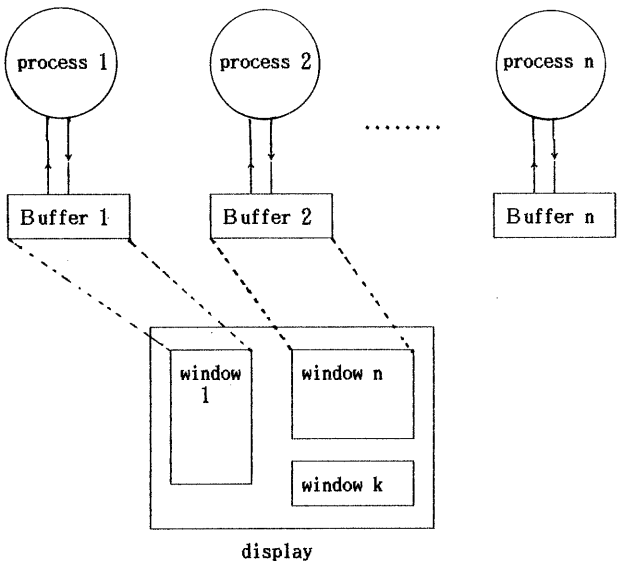

図５　多重ウィンドウによるユーザインタフェース

6．おわりに

高機能ワークステーションが，真にその特徴を生かした形で使用されるための前提となるＯＳについて，その設計方針について述べた。　従来のＯＳの構成要素であるプロセス，ファイルなどについては，勿論何らかの形で実現されるわけであるが，ここでは，それらの上に更に新しい概念に基づいた管理および制御機構が必要であることを強調した。これらの方針に基づいて設計したＯＳを実際に作成し，それらを評価してみることが，次の課題であると考えている。

参考文献

〔1〕 Alto User's Handbook, Xerox PARC, Sep. 1979

〔2〕 斎藤信男：高機能ワークステーションの応用，情報処理，Feb. 1984 掲載予定

〔3〕 S. R. Bourne, "An Introduction to the Unix Shell," Bell Labratories, N. J. Nov. 1978

〔4〕 Second Compendium on Gandalf Project, Dept. of Computer Science, Carnegie-Mellon Univesity, 1982

〔5〕 L. E. Bonanni and C. A. Salemi, "Source Code Contorol System User's Guide," Bell Laboratories, N. J.

〔6〕 J. Gosling, Unix Emacs, Dept. of Computer Science, Carnegie-Mellon Univesity, 1982

〔7〕 D. M. Ritche and K. Thompson "UNIX Time-Sharing System," CACM 17 No.7 July 1974 pp.365-375

〔8〕 K. Thompson, "Unix Implementation," Unix Documentation, Bell Laboratories, N. J. Nov. 1978