東大センター副システム（VAX-11/780）用 UNIX/32V のソフトウェア・アーキテクチャ

長谷部 紀元 （図書館情報大学）
石田 晴久 （東大大型計算機センター）

1. はじめに

UNIXはベル研究所のRitchieとThompsonによって開発された。元来はミニコン（DEC社製のPDP/11）用の汎用TSSシステムである〔1〕。その簡潔で強力なOSの機能、完備したユーティリティ・プログラム、システム開発用高級言語の存在などの特徴によって早くから研究者の間で注目を集め、計算機科学の教育・研究に広く使用されてきた。近年メインフレームの中型機程度の能力を持つVAX 11/780がPDP/11の上位機種として登場して以来、人工知能研究などさらに広汎な分野での研究に使用されようとしている。またパーソナル・コンピュータの分野でも標準的なモニタとしての評価が高まり、ソフトウェア・ハウスが多数のUNIX仕様のシステムや、個別アプリケーション・プログラムの販売を行うなど、ブームに近い状態になってきている。

UNIXにはベル研究所から公表されているもの、カリフォルニア大学バークレー分校（UCB）が拡張したものなどいくつかの版があり、表1にそれを示す。

表1.

Version名	適合 CPU	適要
LSX	LSI-11+フロッピー	シングルユーザ用
MX(mini-Unix)	PDP-11(LSI-11)+ディスク	メモリ・マネジメントを持たない小規模CPU (小人数用)
UNIX 6-th ed.	PDP-11/40,45(LSI-11/23)	CACMに発表された版
UNIX 7-th ed.	PDP-11/45,70	大規模システム用の拡張版
UNIX/32V	VAX-11/780	7-th ed. とほぼ同一レベル
VM/UNIX	VAX-11/780,750	UCBによるオンデマンド・ページング版

ユニックス全般については既に紹介がなされているので〔12〕、本稿ではVAX-11/780上のUNIX（VM/UNIX）について文献〔2～11〕に依りながら、利用者の側からみたそのアーキテクチャーについて紹介する。

UNIXシステムが高い評価を受けている理由として、単にオペレーティング・システムとしての設計がよいばかりでなく、それを実際に利用した強力なソフトウェア・ツールやアプリケーション・プログラムが極めてよく準備されている点が大きい。

以下では注目すべきソフトウェアについて解説する。なおUNIXでは実行プログラムはその格納されているファイルの名称をコマンド名として起動される。従ってユーティリティ・プログラムとコマンドはほとんど同義語であるが、ここでは上に述べた観点で使い分けている。

2. 一般的ソフトウェア・ツール

2-1 ed と grep

edはUNIXの標準の、強力な文脈参照機能をもったテキスト・エディタである。特徴としては、

① コマンドの対象範囲を行番号の他に文脈（その行が含むべき文字列）で指定できる。

② 文脈や置換すべき文字列の指定に正規表現（regular expression，オート

マトン理論のものを拡張したもの）が使用できる．

③ 指定した文字列（正規表現）を含む全ての行を一括して処理するグローバル機能がある．

などがあげられる。 図1に正規表現（'/'で囲んで使用している）とコマンドの例を示す。 edは"Silent System"と言われるUNIXの中でも特に寡黙で、コマンド，テキストのどちらについても入力勧誘出力を行わない。これは一見不親切のように思えるが、UNIXが全二重通信を行っているため、システムの応答速度を気にせずに「先き打ち」ができる大きな利点がある。 他のシステムに余りみられない特長として、エディット中に任意のunixコマンド（ユーティリティ・プログラム）が使用できる。edの正規表現の記法は文字列指定の手段として強力である他、多くの他のユーティリティ・プログラムと共通であって、UNIXの操作性の向上に大きく役立っている。また少し異る形ではあるが、コマンドのオペランドのファイル名の指定にも使用できる（図1）。grepはedのgコマンド（グローバル）から来た呼び名で、（図1の⑦4行参照）、多数のファイルの中から問題とする文字列を含む行やファイル名を探索することができる。

(1)	/abc/	abcを含む行に移る
	/^a/,/z$/p	aで始まる行からzで終る行までをプリントする
	g/a[01]/d	a0, a1を含む行を全て消去する
	g/a.*c/p	ac, abc, axc, axyc…を含む行をプリントする
(2)	$ rm *	全てのファイルを消去する
	$ cc *.c	.cで終るファイル（Cソース）をコンパイルする
	$ pr a[0-9]	a0～a9までをプリントする
	$ cp a* ../his	aで始まる全てのファイルを隣のhisのディレクトリの下にコピーする．

図1 正規表現を使用したedコマンド(1)とunixコマンド(2)

2－2 vi

edは低速の端末を使用することを前提とするエディターとしては一つの完成品と考えられる。しかし高速回線で接続されたインテリジェントなディスプレイ端末（画面メモリー上での、編集機能を有するもの）が利用できる環境では、それを利用したより強力なエディターが要求されるであろう。viはUCBで開発された、画面上での修正がCPU中のテキストと常に一致している、いわゆる画面エディターである。これはUNIXの全二重通信の機能をフルに利用していて、端末から入力される ないし数文字の列からなるコマンドを解釈しては画面を書き換える。（例えばhはカーソルの右移動，xは1文字消去）。画面修正のためのコントロール機能は当然のことながら端末の機種毎に異るので、この差異を吸収するために機種毎の機能を登録する機能を持っている。この機能ファイルのおかげで多種の安価な端末で画面エディターが利用可能である。viの文字コマンドは、対象の単位が文字、単語、行、パラグラフ、ファンクションなどと多種多様なものがそろっており、多すぎるきらいもある。なお下敷としてほぼedと同じ文脈エディターが組み込まれており、例えばグローバル・コマンドなどが随時使用できる。 UNIXを活用した面白い機能の一つとして、エディット中のテキストの一部を任意のユーティリティ・プログラムに転送し、その結果でテキストを置き換えることができる。
なおUCBで作成されたプログラムにはviと同じ機能ファイルを用いたインテリジェント端末向きプログラムが多数ある．

2-3 diff

diffはテキスト・ファイル間の差異を調べるためのものである。これによって新旧のファイルでどのような追加、削除、変更がされているかを調べることができる。面白いのは指定によって旧ファイルを新ファイルに変換するためのエディットコマンド(edの)が出力できることである。(図2)。これを利用すれば基本のファイルと各版の間の更新コマンド・ファイルだけを保存しておけば、任意の版の導出が可能であり、ファイル保存のスペースが倹約できる。

```
$ diff -e p1.p p2.p
5a
od
.
4d
1c
fro i:= 1 to 10 do
.
```

```
$ cat p1.p
for i:= 1 to 10 do
    if m[i] mod 2 = 0 then
            s := s+1
    else
            s := s-1
```

(P1.Pの内容)

図2 p1.pをp2.pに変えるためのedコマンドの出力
(5行目の次に1行追加、4行目を消去、1行目を変更すればよい)

UNIXでは一個のアプリケーションに属するファイルは一つのファイル・ディレクトリーの下にまとめて置くのが習慣である。この場合のディレクトリはIBM流のOSでの区分編成ファイルに相当すると考えることができる。diffにはこのようなディレクトリ間の差異を一括して調べる機能もある。

2-4 その他 — UNIXのコマンドの強力さ

上には述べなかったが、ファイルの種々の操作(複写、削除、改名、ダンプ、一覧表作成など)のためのコマンドが完備していることは言うまでもない。UNIXの環境がこれらの基本的なツールを使用する上で他のシステムより優れている点として次のようなものがあげられる。

① ファイルの構成が唯一でしかも単純である。UNIXのファイルはバイトの列として構成されており、コードのような物理的な媒体に拘束された概念はない(テキスト・ファイルの「行」はLF(ラインフィード)で終るバイト列であるが、これは処理プログラムの責任で認識する)。このためファイルの構成法(SAM-DAM、レコード形式、レコード長etc.)によってコマンドの選択、オペランドの指定で思い煩らう必要がまったくない。この唯一性はディスク・ファイルと端末ばかりでなく、磁気テープや物理的なディスク・ドライブの入出力にも適用され、また物理的なメモリーの参照までも包含するよう一般化されている。なお入出力の手段としては順次アクセスの他に、任意のバイト・アドレスに位置付けする機能が用意されている。

② ファイルは階層ディレクトリーの下で管理される。関連するファイルをひとつのディレクトリーの下に集めておけば、後で述べるコマンド・インタプリターの機能で一括して処理することが極めて容易となる。なお毎回長い名称を指定する愚を避けるため、作業の足場となるディレクトリを設定する(つまりその部分の指定は不要とする)コマンドが存在する。この足場を他人のディレクトリに移動することも可能であるので、ファイル・システムを介した協同作業が容易である。

③ プロセス間通信もファイル入出力と統一されている。つまりコマンド・インタプリターが設定する情況の下で、プログラムは端末から入出力してい

るつもりで、並行動作する他のプロセスとデータの交換を行うことができる（図3）。このプロセス間通信のインターフェースはpipeと呼ばれる。このpipeを有効に利用するため、UNIXではユーティリティ・プログラムの入出力を「標準の入出力」（これは普通には端末に割り当てられる）にすることが推奨され、そのように書かれたプログラムはfilterと呼ばれる。しかもその方がユーディング"が容易である。

```
(1) $ num text | fold | pr
(2) $ tr "[A-Z]" "[a-z]" <input >output
```

(1) textに行番号をつけ、72文字目で折り曲げ、ページフォーマットで印刷する（|がpipeを表わす）
(2) inputの大文字を小文字に変換してoutputへ格納する。（標準入出力の切り替え）

図3

パイプの効果は論理的には、中間作業ファイルを仲介として順次にプログラムを実行させるのと同じである。違うのは並行処理の結果として最終出力が遅れなしで出てくる点で、これは会話型処理では根本的に重要である。

④ 強力なコマンド・インタープリターのサポートがある。 UNIXのユーティリティプログラムは一般に、複数のファイルをまとめて処理する機能を持つ。従ってコマンドのオペランドはファイル名の並びである。さきにも述べたように、コマンド・インタプリターは正規表現を展開してディレクトリから適合するファイル名を検索し、結果をプログラムに引き渡す（図1）。従って多数のファイルを一括・系統的に処理することが容易である。フィルターに対しては、pipeで継ぐことの他に指定されたファイルを標準入出力に割り当てる機能がある。（図3）。この結果一個のフィルターは労せずして3通り以上の使用法（端末、pipe、ファイル、その他）が可能である。その他に後に述べるような強力なマクロ化機能があり、既存のコマンドから新らしいユーティリティを組み立てることが可能である。

⑤ UNIXのファイル・システム（1） IBM流のOSでは使用すべきファイルの指定の方法が極めて重く（④とまさに逆の情況があるためである）、DDやALLOCATEのコマンドを使いこなすには多大の困難が伴う。これに対してUNIXではそのような手段はまったく必要ない。ファイル・オープンのシステムコールのモードの指定だけで（実行時ルーチンの補助があるものの）、必要ならば新規に作成されて、使用可能となる。またファイルのサイズは自動的に必要なだけの大きさに調節される。そのためにディスクはあらかじめ固定長（512ないし1024B）に分割されており、これを単位として自由に拡張することができる。

⑥ UNIXのファイル・システム（2） ここでは①と③で述べた入出力インタフェースの統一を可能にしているファイル・システムの構成について説明する。UNIXではファイルは、その名称に従って階層ディレクトリをたどって得られるi-nodeと呼ばれるコントロール・ブロックによってその属性が記録されている。通常のファイルではここにファイル実体のディスク上のアドレスが記録されている。これに対して端末やディスク・ドライブ、メモリなどは同様に階層ディレクトリの下に登録されていて、そのi-nodeには通常のファイルと異なるものである旨（special file と称される）の表示があり、対応する入出力ハンドラーの名称が（コーディング"されて）記されている。UNIXのモニターはオープン処理のあとはこの情報をもとに入出力要求を適切なハンドラーへ転送する。

pipeの作成の場合には、階層ディレクトリーには入らない一時的なi-nodeが生成され、その後は同様の経路で入出力要求の転送が行われる。

3. プログラム言語

3-1 C

UNIXの主たるプログラム言語は何といってもCである。Cは元来UNIXの記述を念頭において設計されたものであり、カーネル、Cコンパイラ自体を含めてUNIXシステムの大部分がC言語で記述されている話は有名である。UCB版のVAX用UNIXのカーネルでは、コメントを除くと、全体7945行のうち1015行だけがアセンブラ言語である。他でアセンブラが使用されているのは、システム・コールの実行時ルーチンやlispインタプリターの一部などごく例外的な部分だけである。

図4. C, Pascal, Algol 68 の比較

```
[C]
        for (i=1; i<=10; i++)
                s += m[i] % 2 ? -1 : 1;
[Pascal]
        for i:=1 to 10 do
                if m[i] mod 2 = 0 then
                        s := s+1
                else
                        s := s-1
[Algol 68]
        for i from 1 to 10 do
                s +:= (m[i] mod 2 = 0 | -1 | 1)
        od
```

Cは BCPL から発展した Algol 流の言語であり、現在ではかなり強くタイプ付けされたものとなっている。構造化文のレパートリは PASCAL に匹敵するくらい強力であり、細部の表現力の強さにおいては Algol 68 に似た趣きがある(図4)。本言語は最近ではUNIXとは独立にマイクロコンピュータなどでも使われている。

3-2 Fortran

```
63      if (j-k) 64,64,69       while (j<=k) {
64      jip2 =jip2+ira                  jip2 = jip2+ira
        j    =j+1                       j = j+1
        goto 63                         }
69      if (ind) 73,73,70       if (ind>0) {
70      ipli =(i-1)*ica+1               ipli = (i-1)*ica+1
        if (i-n2) 71,72,72              if (i<n2)
71      ijp  =ijp0                              ijp = ijp0
        goto 73                         else
72      ind  =0                                 ind = 0
73      continue                        }
```

図5. structによるFORTRANからRatforへの変換

Kernighanの名著によって一躍有名となったRatforがあるのは当然であるが、その逆(FortranからRatforへ)を行うStructというプロセッサがある。図5にその実行例を示すが、コントロール・フローの解析をちゃんとやっていると思われ、なかなか強力である。なおFortran77のコンパイラも存在する。

3-3 Pascal

UCBで開発されたのでインタプリターとコンパイラーの両方がある。文法解析はYaccを利用しているが、実に細い文法エラー回復の処置を行っている。例えばbegunのようなミススペルを修正するようなことをする。

3-4　Lisp

これにもUCBで開発されたインタプリターとコンパイラーがある。文法仕様はMaclispを受け付けるようになっている。UCBではこの上に大規模な数式処理言語であるMACSYMAをのせる計画が進行しているという。

3-5　Yacc

Yaccは実用的なパーザー・ジェネレーターであり、LP(1)文法を理解するパーザーを出力する数式印刷プログラム（後で述べるeqn）の作成に用いられたことで有名である。生成規則の矛盾を解決するための優先処理の機能を持っているため、文法の入力が簡単な他、出力の実行速度も速いという。先にも述べたようにUNIXの中で盛んに使用されており、VAXのUNIXではCのコンパイラーさえこれで書かれている。

3-6　Lex

Lexは Yaccのためのターミナル・シンボルのアナライザーの作成を簡単にするよう設計された、正規表現解析プログラムのジェネレーターである。Yaccと独立に使用することももちろん可能であり、図6に使用例を示す。これは入力テキストのうち、訓令式ローマ字として読める音節のみを出力（ECHO）するプログラムである。

図6　lexによるプログラム

```
V       (a|i|u|e|o)
C       (k|g|s|z|t|n|h|b|p|m|r)
CY      ({C}y)
CH      ((kk)|(gg)|(ss)|(zz)|(tt)|(bb)|(pp))
CHY     ({CH}y)
CK      ({C}|{CY}|{CH}|{CHY})

%%
{V}             ECHO;
{CK}{V}         ECHO;
d{1,2}(a|e|o)   ECHO;
y(a|u|o)        ECHO;
wa              ECHO;
n               ECHO;
.               ;
%%
```

3-7　sh

UNIXではコマンド・インタプリターのことをshellと呼ぶ。そのshがここに登場する理由は、その理解するコマンド言語が、高級プログラム言語としての機能を備えているからである。これで先に紹介したコマンドその他を組み合わせると、大抵のことはコマンド言語のみで出来てしまう。なおここで重要なことは、コマンド言語で記述されたものが再び1個のコマンド（shell script）として呼ぶことができ、しかも使用法（オペランドの指定方法）が実行プログラムによるものと全く変わらないことである。図7に構造化文のシンタクスを示す。

```
for name [in word ...] do list done
case word in [pattern [ | pattern ] ... ) list ;;] ... esac
if list then list [elif list then  list] ... [else  list] fi
while list [do list] done
```

図7. shellのシンタクス

詳しい説明は省略するが、listはコマンドまたは';'で区切られたコマンド列でifやwhileはその最後のもののリタンコードをチェックする。[ ]は省略可能な単位、...は同じものを繰り返し書いてよいことを示す。

4. 開発メンテナンス支援ツール

4－1　デバッガー

記号変数でのアクセスができるポストモーテム・インタラクティブ両用のデバッガとしてadbがある。機能は余り特別ではないがUNIXとしての面目は、これがディスク上のファイル・システムの修復に使用できる点にある。つまりメモリー上のプロセスのイメージも、物理的なディスクも共に通常のファイルと同じにアクセスできるからである。他にsdbというシンボリック・デバッガーがある。これはデバック中にソース・ファイルを自由に見ることと、ステートメント単位、ファンクション単位の実行もシンボリックにできる点が特徴で、C、Fortran、Pascalなどに適用できる。sdbや先に述べたvi（画面エディター）を使用すると、プログラムリスティングなどの紙切れはほとんど必要を感じない。問題となるコーディングを目で探す労力や保管・管理の手数を考えるとむしろ邪魔ですらある。

4－2　SCCS と make

SCCSはPWB/UNIXにあるパッケージで、大規模なシステム開発・保守の過程におけるソース・コードの世代管理をするためのものである。各モジュールの世代間の差の情報を蓄積しておき、任意のレベルの全ソース・コードをいつでも再現できる。makeは2で述べたような多様な言語で書かれた多数のソース・ファイルを集成して実行プログラムを生成する作業を容易にするためのものである。あらかじめ用意されるシナリオに従って動作し、必要なコンパイラやプリプロセッサーを自動的に呼び出す機能や、前回の生成作業後に更新されたソース・モジュールのみを処理対象として、生成処理を最小限のものにする機能がある。

5.　文書処理用ツール

5－1　troff, eqn, tbl

これらは写植機用の文書清書ソフトウェアで、Bell研究所ではUNIXのドキュメントは全てこれで作成している（ソーステキストは全体で約10MB）。さらに数冊の単行本を出版していることもよく知られている（例えば参考文献の[3]）。troffは高級なフォーマッターであり、eqnとtblはそのプリプロセッサーで、それぞれ数式と表の印刷に用いられる。これらと双生児的関係にある高級型電動タイプライタ端末用のソフトとしてnroffとneqnがあり、troffと同じ入力を受付ける。図8にその入出例を示す。

図8. neqn, nroffによる数式出力の例（上半分が入力）

```
.na
.EQ I
G(z)~mark = ~ e sup {ln ~G(z) }
.EN
.sp
.EQ I
~~~~~
= prod from k>=1 e sup {S sub k z sup k /k}
.EN
```

$$G(z) = e^{\ln G(z)} \qquad \Pi, \geq \text{は重ねうちによっている}$$

$$= \prod_{k \geq 1} e^{S_k z^k / k}$$

5-2 spell, refer, style, diction

疑しいスペルを小規模の辞書ファイルに照らしてチェックするspellは既によく知られている。referはtroffのプリプロセッサーで、参考文献を簡単な文献検索の機能（キーワードを.[と.]のtroffコマンドでかこんで指定する）で共通のデータ・ベースから取り込む。本稿の参考文献の[2～7]はこれを利用して

作成したものである。style, dictionは読み易い英文を書く補助手段として用意された。前者は統計的な手法で長すぎる単語・文をチェックする。後者は長い語句を短く言い替える可能性を指適する。

6. UNIXの東京大学大型計算機センターにおけるインストール

本センターでは最初はLSI-11でのMini Unix, 次にLSI-11/23での6-th edition, そしてVAX-11/780でのVM/UNIXとインストールしてきた。ここではVAX 11/780で行った作業と発注した問題点について述べる。

① セットアップ ── UNIXのシステムは磁気テープとドキュメントの冊子で配布される。セットアップに関する資料はさすがに手際よくまとめられており、標準構成のハードウェアであれば全く問題なく進めることができる。ディスク・パックのハード的なイニシャライズを含めて、半日強でシングル・ユーザ・モードの運転が可能となった。

② 回線サポート・ハードウェアとソフトウェアのバグ ── 本センターでは32回線の調歩同期端末をサポートするためにdh11/dm11というDMA転送とNCU制御の機能をもったハードウェアを使用している。これに関連してハード・ソフト両者のバグが重なって存在し、その解決にほぼ1ヶ月を要した。

③ NCUのサポート ── 日電製のNCUを使用したが、これはリレー制御のため反応が遅く、セッション終了時のソフトの処理に追随できない。これらの問題の解決のために時間監視の機能をハンドラーに追加した。これには内部テーブルがあふれ、UNIXのコーディング不良と重なってダウンを引き起すというおまけがついた。 ②と併せて考えて見ると、UNIXを電話網から使用する事は以外と少いのかも知れない。

7. おわりに

VAX-11/780のハードウェアはほぼ安定して動いている。本システムは現在、20回線の電話網端末のサポートを含めて24時間運転を行っている。2～3月の実績は実使用者数66人, 運転日数は43日、セッション・タイムの総計は2678時間、1日の最大使用者数は44人であった。情報検索やLISPを使用した研究に利用されている。

〔参考文献〕

[1] D. M. Ritchie and K. Thompson: The UNIX Time-Sharing System, Communications of The ACM, Vol.17, pp.365-375 (1974)

[2] K. Thompson, "UNIX Time-Sharing System: UNIX Implementation," Bell Sys. Tech. J. Vol. 57(6) pp. 1931-1946 (1978).

[3] B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey (1978).

[4] S. R. Bourne, "UNIX Time-Sharing System: The UNIX Shell," Bell Sys. Tech. J. Vol. 57(6) pp. 1971-1990 (1978).

[5] B. W. Kernighan, M. E. Lesk, and J. F. Ossanna, "UNIX Time-Sharing System: Document Preparation," Bell Sys. Tech. J. Vol. 57(6) pp. 2115-2135 (1978).

[6] S. C. Johnson and M. E. Lesk, "UNIX Time-Sharing System: Language Development Tools," Bell Sys. Tech. J. Vol. 57(6) pp. 2155-2175 1978).

[7] T. A. Dolotta, R. C. Haight, and J. R. Mashey, "UNIX Time-Sharing System: The Programmer's Workbench," Bell Sys. Tech. J. Vol. 57(6) pp. 2177-2200 (1978).

[8] UNIX Time-sharing System: UNIX Programmer's Manual, Seventh Edition, Virtual VAX-11 Version, Computer Science Division, Department of Electric Engineering and Computer Science University of California, Berkeley, California (1980)

[9] UNIX Time-sharing System: UNIX Programmer's Manual, Seventh Edition, Volume 2a, Bell Laboratories, Murray Hill, New Jersey (1979)

[10] UNIX Time-sharing System: UNIX Programmer's Manual, Seventh Edition, Volume 2b, Bell Laboratories, Murray Hill, New Jersey (1979)

[11] UNIX Time-sharing System: UNIX Programmer's Manual, Seventh Edition, Volume 2c, Virtual VAX-11 Version, Computer Science Division, Department of Electric Engineering and Computer Science, University of California, Berkeley, California (1980)

[12] 石田晴久：ベル研究所の移植のOS－UNIX, 情報処理, Vol. 18, No. 9, pp. 942～949 (1977)

[13] 長谷部紀元：UNIXの使用経験, 東京大学大型計算機センター年報第9号 pp. 51～58 (1978)