パーソナルコンピューティング環境の一実現方法

土居範久，東方聖朝，田原敏夫，鹿野芳之（慶應義塾大学）
小坂一也（日本ＩＢＭ）

０．はじめに

充実した個人的計算環境を実現するために，１６ビットマイクロコンピュータ６８０００を用いて，単一使用者，多重プロセスで，機能の拡張および変更が容易な個人的計算環境を開発している[１]。

システムの特徴は次の通りである。

（１）システムの全てが高水準言語ＣＯＭＰＬ[１]で記述されている。ＣＯＭＰＬは，構造化言語ＳＩＭＰＬにＳＩＭＬＡ６７のクラス（class）の概念を取り入れて拡張した言語である。

（２）入出力操作がクラスとストリームの概念[２]を用いて設計・製作されている。

（３）ファイルシステムはＯＳ６[３]のものを改良して用いている。

（４）コマンドプロセッサはＵＮＩＸ†のShell相当のものである。

（５）ソフトウェアは「上位の層は使わない」という規則に従った階層構造となっている。

０．１　機器構成

システムの機器構成は図１のとおりである。

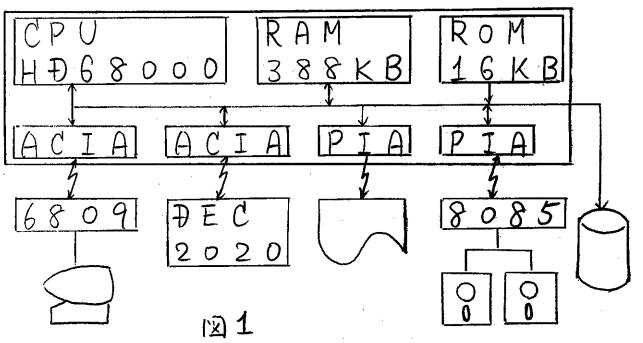

図１

０．２　ソフトウェアの構成

ソフトウェアの概略は図２のとおりである。

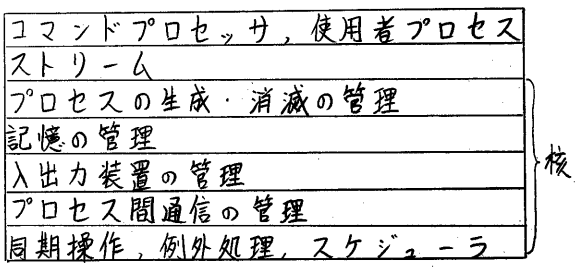

図２

１．ＯＳの核

ＯＳの核は前述のように五層に分かれており，プロセスの管理，記憶の管理および入出力装置の管理はおのおの独立した幾つかのプロセスから成っている。

１．１　プロセスの状態

プロセスの状態には次の七つがある。

Dead：プロセスが存在していない状態

Dormant：クリエイタによって生成中の状態

Ready：走れる状態にあるがＣＰＵが割り当てられていない状態

Waitting：Ｐ命令によって待機している状態

Running：ＣＰＵが割り当てられて走っている状態

Swapped：他プロセスをロードするためにスワップアウトされている状態

Terminated：実行が終了してターミネータによって資源を開放中の状態

†UNIXはベル研究所の登録商標である。

1.2 プロセス管理表

プロセスを管理するプロセス管理表（Process Control Block, PCBと略記）の主な内容は次の通りである。

Pcb$Name (string) プロセス名
Pcb$Stat (int) プロセスの状態
Pcb$Prty (int) 優先度
Pcb$Time (int) 消費時間
Pcb$Ptr (int) 実行待ちリスト，セマフォ列へのポインタ
Pcb$Par (int) 親プロセスのプロセス番号
Pcb$Sem (int) 接続されているセマフォへのポインタ
Pcb$Sp (int) スタックポインタ退避領域

1.3 同期操作

同期操作は計数セマフォを用いて行なっている。P－V命令を不可分な命令として実現するためのLock_Unlock命令には割込みの禁止・解除を用いている。

P-V命令は次の手続きによって実現している。

Initsem(セマフォ，初期値) セマフォの初期設定
Wait(セマフォ) P命令
Signal(セマフォ) V命令

1.4 ディスパッチャ

計時割込みを用いた時間刻み化によってもプロセスを切り換えている。ディスパッチャはP命令および計時割込みによって呼び出され，次に実行すべきプロセスを決定する。

ディスパッチャでは優先度にもとづいた多重レベルリストを実行待ちリストとして用い，空でない最も高い優先度の実行待ちリストの先頭からプロセスを選ぶ。また，常に走るべきプロセスが存在することを保障するためにNullプロセスを置いている。

1.5 プロセス間通信

プロセス間通信にはバッファ方式を用いている。システムに枠バッファと呼ぶメッセージを入れるための固定長の枠の配列を用意し，メッセージを送信する時には，メッセージを枠の大きさに切り，空いている枠に入れ，送付先のメッセージ列の最後につなぐ。受信の際は自分のメッセージ列から先頭のメッセージを受け取る。空いた枠はバッファに戻す。枠はメッセージ本体，送り主のプロセス番号，メッセージの長さ，種類，メッセージ列につなぐポインタで構成されている。メッセージ列は郵便受け単位に作る。

プロセス間通信を行うための手続きは次のとおりである。

Getbox(プロセス王冠) 郵便受けを一つ取り出す
Freebox(郵便受け番号) 郵便受けを戻す
Sendmsg(プロセス王冠，郵便受け番号，種別，メッセージ) メッセージを送信
Rcvmsg(プロセス王冠，郵便受け番号，種別，メッセージ) メッセージを受信

1.6 入出力装置の管理

入出力装置は，装置の種類ごとに個別に用意したプロセスで管理しており，現在，直列回線，並列回線，ディスクの3種類がある。各管理プロセスは，他のプロセスから入力要求メッセージを受け取ると，機器から入力し，それをメッセー

ジて要求を出したプロセスに送る。出力データのメッセージを受け取ると機器へ出力する。その他機器の特性に応じた特殊なメッセージを受け取って処理を行う。使用者が，入出力管理プロセスを通して，入出力装置を扱うクラスおよび手続きは次のとおりである。

直列回線	Ttyio	ストリーム入出力をするためのクラス
	Tstat	回線・端末の状態を得る
	Tmode	回線・端末のモードを設定する
並列回線	Lptout	ストリーム出力をするためのクラス
	Lstat	プリンタの状態を得る
	Lmode	プリンタのモードを設定する
ディスク	Diskio	ストリーム入出力をするためのクラス
	Format	ディスクをフォーマッティングする
	Dstat	ディスクドライブコントローラの状態を得る
	Dmode	ディスクドライブコントローラのモードを設定する

1．7　一次記憶の管理

一次記憶はメモリマネージャとスワッパの二つのプロセスによって管理している。メモリマネージャは空いている主記憶領域を自由リストを用いて管理しており，要求が来ると自由リストを探して適当な領域を渡す。自由リストの管理には初適合アルゴリズム（FirstFitAlgorithm）を用いている。要求された大きさよりも大きい領域がなければ，スワッパを呼び出し適当なプロセスをスワップアウトして領域を確保する。プロセスが終了すると，それが使用していた領域は自由リストに戻す。

1．8　二次記憶の管理

二次記憶の管理はファイラで行う。ファイラはファイルの入出力要求などのメッセージを受け取り，ディスクハンドラを介して入出力を行い，要求を出したプロセスに結果を返す（3．参照）。

1．9　プロセスの生成・消滅の管理

プロセスの生成はクリエータとローダで行い，消滅はターミネータで行う。

クリエータはプロセスの生成要求を受け取ると，ＰＣＢを初期設定し，必要な主記憶をメモリマネージャからもらい，ローダを介してプログラムをロードする。実行開始要求を受け取るとプロセスを実行待ちリストに挿入する。

ローダはクリエータから記憶位置とロードすべきファイル番号を受け取り，プログラムを主記憶にロードする。プログラムがシステムの入口名を参照している場合にはつなぎをとる。

ターミネータはプロセス終了のメッセージを受け取ると，プロセスをTerminated状態にし，メモリマネージャ等の資源管理プロセスに資源開放メッセージを送る。開放終了後プロセスをHead状態にする。

使用者がプロセスの生成・消滅を行うための手続きは次の通りである。

Create	プロセスを生成しロードする
Act	プロセスを起動する
Abort	そのプロセスの実行を異常終了する
Term	そのプロセスの実行を正常終了する
Kill	他のプロセスの実行を異常終了する

2. ストリーム

ストリーム（Stream）は情報を伝達するための抽象データである。

システムでは，入出力をストリームとして統一的に扱う事によって装置独立を実現している。ストリームもCOMPLで記述されており，クラスによって実現している。通常，ストリームは，入力ストリーム（Input Stream）または出力ストリーム（Output Stream）であるが，ファイルを読みながら更新するような時は入出力を共に行うことができる。このようなストリームは双方向ストリーム（Bilateral Stream）という。

2.1 基本操作

ストリームは入出力されるデータ列とそれに適用される関数や手続きから成る。入力ストリームに適用する基本的な関数はNextであり，その関数値はそのストリームの次のデータである。たとえば，「端末から入力された文字のストリーム」Ttyinに対してNextを作用させる時は，

X := Ttyin.Next

と記述でき，この文を実行するとXには端末から入力された文字が代入される。

出力ストリームに適用する基本的な手続きはOutであり，これはストリームの最後にデータを付け加えるものである。たとえば「端末に出力される文字」から成るストリームTtyoutに対して文字Cを出力する時は，

Ttyout.Out(C)

と記述する。これを実行するとCの値が端末に表示される。したがって，

Ttyout.Out(Ttyin.Next)

という文は端末に対してエコーバックを行うことになる。

ストリームに対する基本操作としては，この他に次のものがある。

Endof	入力ストリームの終了判定
Reset	ストリームの再初期設定
Close	ストリームの使用を終了して後始末をする
Putback	入力ストリームの先頭にデータを戻す

ここで，ResetおよびCloseはすべてのストリームに関して同じ動作をするとは限らない。たとえば，ファイルを扱うストリームでは，ファイルのRewindとCloseに相当する動作を行うが，端末に対するストリームでは何もしない。

Putbackは先読みしたデータをストリームに戻すの用いるための手続きである。たとえば，

Stream.Putback(X)

を実行したあとで，このStreamにNextを作用させた結果はPutbackを実行した時のXの値になる。なお，読み込んだデータ以外をPutbackすることもできるし，Putbackを行う回数の制限もない。

2.2 ストリーム関数

ストリームは，代入することも引数として渡すことも関数値とすることもできる。ストリームはストリーム関数（Stream Function）によって生成することができる。ストリーム関数は，ストリームを引数としてとり，新しいストリームを返すクラスである。たとえば，元のストリームから空白を除いた文字から成るストリームを生成するクラスRemove Spaceがあるものとし，

S1 := new Remove Space(Ttyin)

S2 :- new RemoveSpace(Filein)

を行うと，S1には「端末から入力された空白以外の文字から成るストリーム」が，S2には「ファイルから入力された空白以外の文字から成るストリーム」が代入される。これらにNextを作用させると，どちらも空白以外の文字が得られる。このようなストリーム関数を用いることによって，入出力に対する前処理や後処理を容易に行うことができる。

2.3 ファストストリーム

自動的にバッファリングを行う機能をもつストリームがあり，これをファストストリーム（Fast Stream）という。これに対して，今まで述べてきたようなストリームはスローストリーム（SlowStream）という。ファストストリームは，ファイルの入出力のように，原始ストリームと結果としてのストリームの間で大きさが異る時に用いる。ファストストリームはクラスBufferを用いて実現している。

2.4 ストリームの実現

ストリームはCOMPLのクラスを用いて記述されている。COMPLには，整数，文字および文字列のデータ型があるので上記の基本操作は型ごとにある。クラスStreamは図3のとおりである。ここで，Streamの引数Modeは入出力の方向を指定するものであり，Stream型の変数SubはPutbackされたデータをクラスインスタンスとして保持するために用いるものであり，Buffer型の変数IbufおよびObufはファストストリームにおいて用いられるバッファである。IbsizeおよびObsizeはそれぞれ入出力ストリームにおけるバッファの大きさであり，Inner文によって実行されるストリーム関数の本体によって設定される。Bbsizeは双方向ストリームにおいてバッファを共用する時に用いるものである。Virtual規制子のついた手続きはサブクラスにおいて実際の操作が定義されている。

2.5 プロセス間のストリーム

ファイルの入出力や他のプロセスとの通信に用いるストリームはメッセージ通信によって実現している。これらのストリームには次のものがある。

Fileio ファイル入出力
Ttyio 端末（直列回線）の入出力
Lptout プリンタ出力
Diskio ディスク入出力
Prodio プロセス間通信

```
class Stream(int Mode)
inst(Stream) Sub
inst(Buffer) Ibuf, Obuf
int Ibsize, Obsize, Bbsize
virtual char func Pnextc
virtual proc Poutc(char)
char func Nextc
if Sub.Ibsize then
  return(Ibuf.Pnextc)
else
  return(Sub.Pnextc)
end
proc Putbackc(char C)
  Sub :- new Putback(Mode, Sub, C)
proc Outc(char C)
if Obsize then
  call Obuf.Poutc(C)
else
  call Poutc(C)
end
int func Endof
  ----------
Sub :- this Stream
inner
if ibsize then
  ibuf :- new Buffer(Input, Sub, Ibsize)
end
  --------
endclass
```

図3

また，機器独立を保つために下記のようなストリーム名から上記のストリームを生成する関数Openがある。

FD0：mydir/myfile.txt	ファイル	LP：	プリンタ
TT1：	端末	FD1：	ディスク

2．6　標準入出力ストリーム

使用者がプログラムを書く時には標準入出力ストリームを用いることができる。標準入出力ストリームは，プログラムが走る前にコマンドプロセッサによって生成され，実行終了後Closeされるので，使用者はプログラム中で自由に使用することができる。標準ストリームの入出力対象の決定はプログラムの実行時にコマンドプロセッサによって行う。標準ストリームには次の四つがある。

$sysin	標準入力ストリーム	$sysgen	標準オブジェクトストリーム
$sysout	標準出力ストリーム	$syserr	標準エラーストリーム

3．ファイルシステム

ファイルシステムはOS6のファイルシステムを改良したものを用いている。

3．1　ファイルの構造

ファイルの構造は図4のとおりである。ファイルはヘディングと本体から成る。ヘディングはファイルの大きさ，先頭および最終ページへのポインタ，作成日等のファイルの状態を記録した部分であり，全ファイルのヘディングはヘディングファイルと呼ばれる特殊なファイルにすべて収められている。ファイル本体は512語のページを双方向ポインタでつないだ構造をしており，1語目にはページ番号とMFLインデクスが，2語目には前後のページへのポインタが入っている。

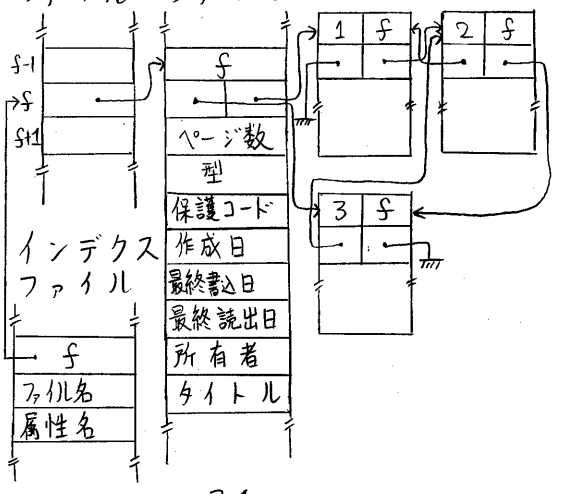

図4

すべてのファイルにはMFLインデクスと呼ぶ一連番号が割り当ててあり，この番号を用いてファイルを一意に定めることができる。またMFLファイルと呼ぶファイルがあり，このファイルをMFLインデクスで索引すると，ヘディングファイルの物理ページと語数が得られる。これによってヘディングを読み込むことができる。

3．2　システム用ファイルと用途

1）インデクスファイル：ファイル名からMFLインデクスを得るためのファイルでファイル名とMFLインデクスの組を並べたもの。ファイルのディレクトリに相当し，ルートディレクトリに相当するシステムインデクスファイルが必ず存在する。

2）MFLファイル：ファイルのMFLインデクスの順にそのファイルのヘディングのある物理ページと語数を並べたもの。

3）ヘディングファイル：全ファイルのヘディングを収めたもの。

4）自由ページリスト・ディスク中の未使用ページのページ番号を並べたもの。
ディスクの空き領域の管理に用いられる。

3．3 ファイルのアクセス

インデクスファイルに新しいインデクスファイルを登録していくことによって階層的なディレクトリ構造をとることができる。ファイル名をもとにそのファイルをアクセスするには次のようにすればよい。たとえば，/Mydir/Subdir/Nameで表わされるファイルを考える。まず，システムインデクス中でMydirというインデクスファイルを探し，インデクスファイルMydir中でSubdirというファイルを探す。最後にSubdirインデクスファイル中でファイルNameを探し，そのMFLインデクスを得る。このMFLインデクスによってMFLファイルを索引し，ヘディングのあるページと語を得る。ヘディングを読み込むと，その中にファイル本体を指すポインタがあり，それによってファイル本体へのアクセスができる。

もちろん，この様なファイルのアクセスはファイラで行うので，使用者は次の様な手続きを用いることによってファイルを扱うことができる。

Fileio	ストリーム入出力を行うクラス
Lookup	ファイル名からMFLインデクスを得る
Fstat	ファイルの状態を得る
Fmode	ファイルの状態を変更する
Rename	ファイル名を変更する
Crefile	ファイルを作成する
Delete	ファイルを消去する
Credir	インデクスファイルを作成する
Deldir	インデクスファイルを消去する
Move	ファイルを別のディレクトリへ移す

4．コマンドプロセッサ

コマンドプロセッサは使用者の入力したコマンドを受け取り解釈・実行する，使用者とOSとの間のインタフェースの役割を果すプロセスである。コマンドプロセッサの基本的な機能は'Load-Go-Loop'と呼ぶ，指定されたファイルをロードし，プロセスとして起動する動作を繰り返すことである。

コマンドプロセッサの特徴は次のとおりである。

1）独立した一つのプロセスとした事で，修正変更が容易である。
2）コマンドで指定されたストリームを標準ストリームに割り当てる機能を持ち，実行時にプログラムの入出力の切換えができる。
3）上記のストリームとして他のプロセスも指定できるので，パイプラインの機能をもつ。
4）複数のコマンドを同時に実行できる。
5）コマンドを任意のストリームから読んで実行する，いわゆるコマンドファイルを拡張した機能がある。
6）コマンドのオプションや引数をプロセスに渡す。その際に引数の中のワイルドカード文字の置き換えをすることができる。
7）if，whileなどのコマンドによりプログラムの実行を制御することができ，プログラムが書ける。

8）コマンドやファイルを探す際のデフォルトディレクトリが指定できる。
9）プロセスの出力を変数に代入したり，入力を変数から読み込むことができる。
10）ログイン，ログアウト時にコマンドファイルを実行することができる。
11）コマンド名，ストリーム名等に別名がつけられる。

4．1　コマンドの文法

コマンドの構文は図5のとおりである。";"は逐次実行を"&"は並列実行を"!"は並列実行および前のコマンドの標準出力を後ろのコマンドの標準出力につなぐパイプラインを表す。引数中の<,>,=>,->は標準入出力ストリームの切換えを示す。

```
Command ::= Pricom | Pricom ; Command
            Pricom & Command | Pricom ! Command
Pricom  ::= (Command) | Com [Option] [Arg]*
Com     ::= filename | buitin
Option  ::= 空白を含まない文字列 | "文字列"
Arg     ::= <Stream | >Stream | =>Stream | ->Stream
            区切り文字を含まない文字列 | "文字列"
Stream  ::= filename | device name | process | variable
Builtin ::= login | logout | term | abort | kill |
            if | while | exit | define | set
```

図5

4．2　コマンドプロセッサの実現

コマンドプロセッサ全体の構成は図6のとおりである。ここでGetcomはコマンドを区切りまで読み込む関数であり，Execがコマンドの解釈・実行を行うための本体である。Execはコマンドを解釈し，ロード実行を行うが，コマンドの区切りが"&"であればコマンドの終了を待たずに次のコマンドの解釈・実行を行い，それ以外の時は終了を待つ。

```
proc CP
  repeat
    if login then ログインファイルの実行 end
    repeat
      Command := Getcom(Delim)
      call Exec(Command, Delim)
    until login, logout, abort, term end
    if logout then ログアウトファイルの実行 end
  until logout, abort, term end
endproc
```

図6

4．3　入出力の切換え

コマンド中に"<"（入力ストリームの指定），">"（出力ストリームの指定），"=>"（オブジェクトストリームの指定），"->"（エラーストリームの指定）がある時には，コマンドプロセサはストリーム名を起動されるコマンドプロセスに渡す。これらはコマンドプロセスの初期設定ルーチンでOpenし，おのおのコマンドプロセスのSysin～Syserrストリームに割り当てる。また指定がなかった時には，'TTO:'を渡す。パイプラインが用いられた時には"!"の両側のプロセスをCreateし，プロセス通信用の郵便受けを作ってコマンドプロセスに渡す。

5．おわりに

システムは現在開発中であり，ここで述べたことの中にはアイデアだけを述べたものもある。端末のポインティングデバイスとしてはマウスを用いる予定で，現在，6809に接続しているところである。

［参考文献］

［1］小坂一也，"16ビットマイクロコンピュータ用オペレーティングシステムに関する研究"，修士論文，慶應義塾大学工学研究科（1982）．

［2］土居範久，小坂一也，他，"クラスによるストリームの実現について"，第24回プログラミングシンポジウム報告集（1983）．

［3］Stoy, J. and Strachey, C., "OS6 An Operating System For A Small Computer", Oxford University Computing Laboratory, Programming Research Group,（1972）．