マイクロコンピュータ 28－3
（1983. 9. 26）

α－DOSとその設計思想

菊池秀明
（アドテックインターラプト大阪）

1. はじめに

当社は近い将来マイクロコンピュターの主流になるであろうと思われるモトローラ社のMC68000の研究をつづけてきた。

その中で標準となるOSの選定をどうするかが大きな問題の一つであった。MC68000用のOSとしてはUNIXが最もふさわしいと思われたが，現在のパーソナルコンピュータのように多くのユーザが使用可能にするには，UNIXの必要とするシステムが大きくなりすぎるだけでなく，価格的にも一般のユーザにはあまりに高価なものであった。

また他に68000用としてふさわしいOSがなかったため，OSから開発することに決定した。

2. 設計方針

OSの開発にあたって以下のように決定した。

1) UNIXの豊富なユーティリティを利用できるようにシステムインターフェイスの互換性をもたせる。（文献1）

現在UNIX用のC言語で書かれたユーティリティの量は莫大なもので、それらを利用可能にすることは非常に有用である。

2) フロッピーディスクベースで可動にする。

現在のパーソナルコンピュータから将来のスーパーパーソナルコンピュータまでフロッピーベースのシステムが主流になると思われる。そのために，フロッピーベースで動作することを必要とした。

3) 階層ファイルシステムとする。（文献4,5）

将来の大容量ファイル時代に対応するために階層ファイルシステムとした。

4) 将来のマルチタスク，マルチユーザシステムへの拡張性を考慮する。

特にマルチタスクは現在、パーソナルコンピュータにも強く要求されており，それらへの拡張は十分に考慮されるべきである。したがって1)との関係を含めて次のようにした。

＊ ファイルのプロテクト機能を充実させる。
＊ 時系列の管理を充分に行なう。（文献5）

5) C言語で記述する。

他のシステムへの移植性を重視し、また1)とも深くかかわっている。

6) UNIXとコマンドレベルでのコンパチビリティーをもたせる。（文献2,3）

将来さらに普及するであろうUNIXとコマンドレベルでコンパチブルにすることは，OSごとにちがうコマンドを覚えることを強いられるユーザにとっては、たいへんありがたいことである。

したがって，UNIXのコマンド体系の良悪は別として、あえて準拠している。

7) 入出力のリダイレクションを可能にする。

標準入出力の考え方はユーティリティの作成時に有効である。

3. ファイルシステムの構造

α－ＤＯＳでは、トラック、セクター等の物理的表現を使ってはいず、すべて、シリアルナンバーのついたブロック、として扱われる。ブロックからトラック、セクターへの割付はより下位のＢＩＯＳルーチンで行なわれる。

１つのファイルの中にはブートストラップローダ用の領域、スーパーブロックと呼ばれるブロック、inodeと呼ばれるブロック、データブロック、が含まれる。(図１)　以下順に説明する。

1) ブートストラップローダ領域、

ブートストラップローダのいるところで、8kbite；16ブロック用意している。

この中には、α－ＤＯＳ識別データも含まれる。

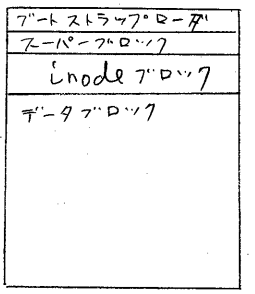
2) スーパーブロック

ファイルシステムの大きさ、inodeの数、及び空ブロック等を管理するブロックである。

フロッピーディスクの場合は１ブロックである。

この中には次のような内容がいっている。

図．１(ファイルシステム)



＊inodeの数　(16bit)　(イニシャル時設定)
＊ブロックの総数(16bit)　(ミニフロッピーの場合 512bite×640)
＊フリーブロックのナンバー　30ヶ分のテーブル.
＊前記テーブルへのポインタ(空ブロックの先頭を示す。)
＊フリーinodeのナンバー　30ヶ分のテーブル.
＊前記テーブルへのポインタ.
＊マルチタスク用フリーブロックロックフラグ.
＊マルチタスク用フリーinodeロックフラグ.
＊スーパーブロックの最終更新日時.
＊フリーブロックの総数.
＊フリーinodeの総数.
＊ファイルシステム名.
＊inodeのサーキュラーサーチのスタートポイント.
＊使用、未使用を示すビットマップのサイズ.

3) inode

データブロックを管理するブロックである。(32bite 16個 / 1block)

この中には次のような情報が含まれる。

a. ファイルモードタイプ　(ディレクトリか一般ファイルか)(パーミッション)
　. パーミッションはユーザー、グループ、全体のREAD、WRITE EXCUTEの許否.
b. リンク数.(１つのinodeにいくつの名前がつけられているかを示す。)
c. オーナーの名前のＩＤ.
d. オーナーのグループのＩＤ.
e. バイト数.(実体の総バイト数).
f. 実体へのポインタブロックへのポインタ.
g. 最終アクセス時間.
h. 最終更新時間.
i. 生成時間.(これらは、1978年1月1日午前0時0分を起点として32bitの秒値)

4). データブロック

実際の各種データが記録されるブロックである。
前述のポインタブロックや、ディレクトリもこのブロックに属する。
以下特殊なデータブロックを説明する。

＊ポインタブロック

ファイルのポインタブロックはinodeによって指される唯一のブロックで、実体へのポインタはこのブロックの中に入っている。ポインタは16bitsの符号なし整数として扱われ、1ブロックに256まで入る。最後のポインタは、次のポインタブロックを指してリンクを構成する。

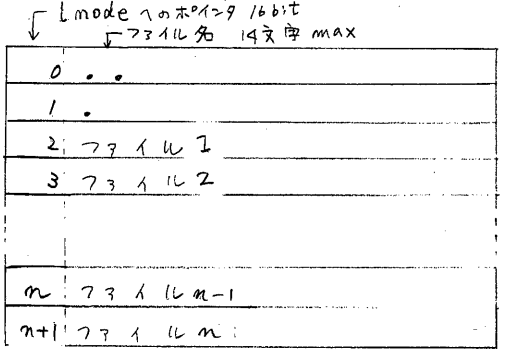
＊ディレクトリ.

図2に示すようにディレクトリはinodeナンバーとファイル名のみが記録されている。
このため、1つのinodeが1つ以上の名前をもつことができる。

mkdir命令などでディレクトリが新たに生成されると、まず..というファイル名で親のディレクトリのi--nodeを指すように記録され、親のディレクトリのリンク数を1つ増す。

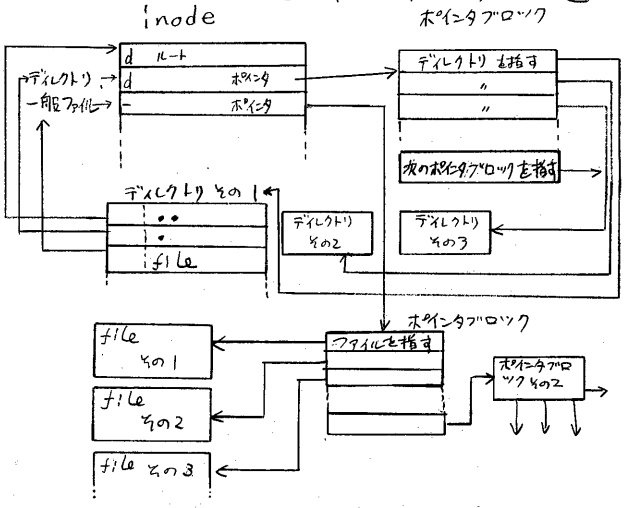
次に自分自身のinodeを指している.というファイル名を生成する。そして次からが実際にそのディレクトリーに属するファイル名とinodeナンバーが入る領域である。

図2. ディレクトリ.



これらの構造で階層ディレクトリを実現している。(図3)

図3. 全体のファイル構造



5) ルートディレクトリ.

交換可能なフロッピーディスクベースでの動作を可能にするために、ルートディレクトリ上に仮想的に各周辺がマウントされている。

したがって使用されるフロッピーディスクドライブや、その他の外部記憶装置、及び端末、プリンタ等の周辺は、このディレクトリ内にあるファイルとして、同様に取扱われる。また特別なディレクトリで、binという名前で、ルートディレクトリに存在するものは、レジデントでないコマンドが収納されており、実行コマンドファイルとして、どのカレントディレクトリにいる場合でも使用できる。

4、α-DOSシステムインターフェイス

α-DOSのシステムインターフェイスは，UNIXとの互換性を考慮して設計している。特にC言語で書かれた豊富なユーティリティをインプリメント可能にするために、C言語レベルでの入出力関数をサポートできるようにしている。

すべての入出力はファイルの読み書きによって行なわれ、前述のようにすべて周辺も同様にファイルとして扱われる。(文献1)

ファイルをOPENするとシステムはユーザにファイル記述子（16bitの整数で、エラーのときには負の値になる。以下fdとする。）を返す。したがってユーザはファイルをアクセスするには常にこのfdによってファイルを識別する。

fdの0は標準入力、1は標準出力，2は標準エラー出力に割当てられており特にこれらは実行時にコマンドインタプリタによって切換を行なうことが可能になっている。

さて，各システムインターフェイスのコールはMC68000のTRAPハンドラーを用いて行なうようになっている。

以下順に説明するが，文中D0～D2　A0，A1，は各々MC68000の汎用レジスタを示し，TRAPはMC68000のトラップ命令，DC.W (値)はTRAPにつづくイミディエイトデータを示す。アセンブラニーモニックの表記は、モトローラアセンブラリファレンスマニュアル(文献6)に基づいている。

1) ファイルのOPEN〔ファイルを操作する場合，まず最初に実行しfdを得る〕
   A0に"file name + NULL"の入っているアドレスの先頭，<32bit>
   D0に　入力，0、　出力，1　　更新，2　<16bit int>
   をセットし，以下を実行する。
   ```
   TRAP    #0
   DC.W    1
   ```
   D1にfdがセットされて返る。（負の値はエラー）

2) ファイルのCLOSE〔ファイル操作を終了した後，fdを解放する。〕
   D0にfdをセットし、以下を実行する。
   ```
   TRAP    #0
   DC.W    2
   ```
   D1に正常終了のときには0，エラーのときには負の値がセットされる。

3) ファイルのREAD　〔任意のバイト数の読みこみ〕
   D0にfd
   A0に読みこむメモリアドレスの先頭<32bit>
   D1に読みこむバイト数<32bit>
   をセットし、以下を実行する。
   ```
   TRAP    #0
   DC.W    3
   ```
   D1に正常終了のときには0、エラーのときには負の値がセットされる。

4) ファイルのWRITE〔任意のバイト数の書き込み〕
   D0，A0，D1にREADと同様にセットし以下を実行する。
   ```
   TRAP    #0
   DC.W    4
   ```
   D1に正常終了のときには0、エラーのときには負の値がセットされる。

5) LSEEK〔ファイルの中のどこから読み書きするかの決定〕

前述のREAD、WRITEは、以前アクセスしたところの直後からアクセスされるが、このLSEEKを使うことにより、実際に読み書きすることなく、ファイルの中を自由に動きまわれる。したがってファイルをランダムアクセスすることが可能になり、ファイルを大きな配列としても扱うことが可能になる。

D0にfd
D1にオフセット<サイズ32bit>ORGに示される位置からの変位
D2にORG <16bit> ファイルの最初、現在位置、終り、をそれぞれ0、1、2で示す。

をセットし、以下を実行する。

```
TRAP    #0
DC.W    5
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

6) ファイルの生成〔新たなファイルの登録〕

Aφに"file name+NULL"のはいっているアドレスの先頭<32bit>
Dφにパーミッション(inodeのところで説明)
をセットし以下を実行する。

```
TRAP    #0
DC.W    6
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

7) リムーブ〔ファイル名の抹消〕

リンクエントリが1つの場合には、ファイルそのものも抹消する。しかし、リンクエントリが複数の場合には、ファイルを残して、ファイルネームのみを抹消する。

Aφに"file name+NULL"のはいっているアドレスの先頭<32bit>
をセットし以下を実行する。

```
TRAP    #0
DC.W    7
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

8) メイクディレクトリ〔現在のディレクトリの下に新たにディレクトリを生成〕

Aφに"directory name+NULL"のはいっているアドレスの先頭.
Dφにパーミッション,
をセットし以下を実行する。

```
TRAP    #0
DC.W    8
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

9) リムーブディレクトリ〔ディレクトリの抹消〕

ディレクトリを抹消するが、ディレクトリの下にファイルが存在する場合にはエラー終了となる。

Aφに"directory name+NULL"のはいっているアドレスの先頭をセット

```
TRAP    #0
DC.W    9
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

10) リンク〔file と file1 のリンク〕
2つのファイルを別の名前で登録する。
Aφにリンクされる"file name+NULL"の先頭アドレス
A1にリンクする"file1 name+NULL"の先頭アドレス
をセットし以下を実行する。

```
TRAP    #0
DC.W    10
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

11) チェンジディレクトリ〔カレントディレクトリの移動〕
Aφに"directory name+NULL"のはいっているアドレスの先頭.
をセットし以下を実行する

```
TRAP    #0
DC.W    11
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

12) チェンジモード〔ファイルのパーミッションの変更〕
Aφに"name+NULL"のはいっているアドレスの先頭
Dφにパーミッション
をセットし以下を実行する。

```
TRAP    #0
DC.W    12
```

D1に正常終了のときには0、エラーのときには負の値がセットされる。

13) ゲットモード〔パーミッションの読みこみ〕
Aφに"name+NULL"のはいっているアドレスの先頭
をセットし以下を実行する。

```
TRAP    #0
DC.W    13
```

D1にパーミッションが返る。

このように、これらのシステムインターフェースは、Cの低水準入出力関数に基づいて設計されている。
したがって、a-DOSではC言語は完全にサポート可能である。
以下READを例にして、Cのインラインで書かれたものの一例を示す。

```
long    read(fd, buf, n)
        int fd;     /* ファイルディスクリプタ */
        char *buf;  /* ファイルネームのポインタ */
        long n;     /* byte数 */
{
asm {
        MOVE.W  -2(A6),Dφ
        MOVE.L  -6(A6),Aφ
        MOVE.L  -10(A6),D1
        MOVEM.L A5/A6,-(A7)
        TRAP    #0
        DC.W    3
        MOVEM.L (A7)+,A5/A6
        MOVE.L  D1,-10(A6)
}
    return(n);
}
```

5, コマンドインタプリタの機能. (シェル) (文献2,3)

ユーザとシステムとのやりとりは必ずこのコマンドインタプリタを介して行なわれる。

コマンドインタプリタの主な機能は、命令の解釈と実行, 入出力のリダイレクション, メタキャラクタの管理等である。

常駐コマンドでない場合は, コマンド用の特定のディレクトリ (α-DOSではbin) をサーチし実行する.

現在サポートしている超特殊文字について以下説明する。

1) ; 連続して実行するコマンドの区切り.
いわゆるコマンドとコマンドの区切りで, 前から実行される。

2) < 入力のリダイレクション.
標準入力を何に割当てるかの切換コマンドで, 省略時は標準入力としてキーボード (コンソール) に設定される。
あらゆるコマンドはこれによって入力をキーボードだけでなく、他の機器にも割当可能である。

3) > 出力のリダイレクション.
標準出力を何に割当てるかの切換コマンドで、省略時は標準出力としてモニターTV (コンソール) に設定される。あらゆるコマンドはこれによって出力を他の機器にも割当て可能である。

4) / パス名の各要素の区切り,
階層ファイルの各パス名の区切りに使われる。

5) * 任意の文字列に対応する。

6) ? 任意の1文字に対応する。

7) [ ] [ ] 内の任意の1文字に対応する。

将来、サポートするものをさらに増やしてゆく予定である。
特にマルチタスク化で必要となるものを以下に示す。

1) | パイプ. あるコマンドの出力を次のコマンドに引渡す。

2) & バックグランドコマンド。

尚、参考までに現在サポートされている標準コマンドを以下に示す。

pwd, ls, cd, mkdir, rmdir, chmod, cat, cp,

mv, rm, date, od, cal, asm, aed, dc, udc 他,

6、主なユーティリティー

α-DOSのユーティリティーもすべてC言語で記述されている。

1) アセンブラ(文献6)

モトローラ社のニーモニックに準処したアブソリュートアセンブラであって実行形式のファイルが直接生成される。

将来リロケータブルマクロアセンブラへの拡張を予定している。

このアセンブラの特徴は、参照ラベルの方向にかかわらず、パス1の中で複数回チェックしているため以下のような記述が許される。

```
ONE   EQU   TWO-ONE
TWO   EQU   THREE-TWO
THREE EQU   3
```

ただしあまり多用するとパス1に時間を要する。

また特にMC68000で、オブジェクトが長くなりがちであるアブソリュート系の命令に対しても可能なかぎり短くなるようにしている。

たとえば、

```
        ORG     $1000
LABEL   MOVE.W  LABEL1,LABEL2
        JMP     LABEL3,
LABEL3  TRAP    #0
        DC.W    1
        ORG     $2000
LABEL2  DS.W    1
LABEL1  DC.W    $100
```

などの場合、LABEL1、LABEL2、LABEL3はすべてWORDのオペランドであるからMOVE.WやJMPはオペランドタイプがWORDになるようにアセンブルされる。このとき1回目のパス1の中では各々不定であるため、そのままパス2を実行するとPHASE ERRをおこす。そのためこれらのPHASE ERRがなくなるまで、数回にわたってラベルテーブルを作りなおしている。

ブランチ系のLONGとSHORTについてもこれらを行なうことは可能であるが、プログラム中で多く出てくるためアセンブル時間が長くかかりすぎるので割愛した。

エラー処理のNOP生成におけるPHASEのずれや、複雑な組合せのラベル参照をもつプログラムにおけるアセンブル時間が、いわゆる2パスアセンブラに比較して長いという問題はあるが、最終的にはブランチ系も含めたオブジェクトサイズの最適化を高速に行なえるよう研究中である。

2)、スクリーンエディタ

いわゆる汎用のスクリーンエディタである。将来マルチウィンドウ化を予定しているが、スクロールの速度等を高速化するためには、コンソール自体の特性を生かせた設計が必要で、当面はテルミー11用で実現することになると思われる。

3)、その他、カレンダー、ファイルコンペア、ファイルダンプ等がある。

7. ハードウェア

現在、富士通社パーソナルコンピュータFM-11上でα-DOSの走る68008カードを製作している。

主な特徴を以下に示す。

1)、パーソナルコンピュータの上で68000のソフトウェアが走る。

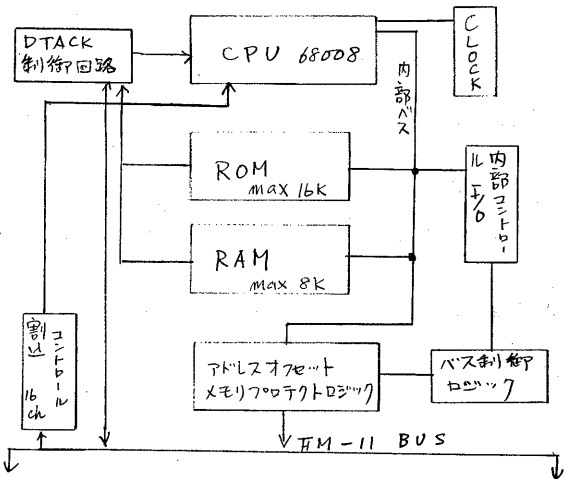
2)、8088、Z80、6809など他のCPUとすべて同居したシステムを構成できる。また68008から他のCPUへ制御を移したり、他のCPUから68008へ制御を返したりすることができるようにバス制御回路を内蔵している。

3)、ROM、RAMを内蔵し、他のCPUにバスを開放している間も68008は、カード内でローカルに命令を実行(並行処理)できる。

4)、将来のマルチタスク、マルチユーザに対応すべく、メモリアドレスのハードウェア的offset機能、メモリライトプロテクト機能をもっている。

ブロックダイアグラムを図4に示す。

図4 ブロックダイアグラム



8、将来への展望と問題

マルチタスク、マルチユーザに関しては、メモリマネジメントユニットや、他のハードウェアの設計を含めてのソフトウェアになるため、研究の余地をもっているが、オペレーティングシステム自体は、設計時点から念頭においているため、完成は時間の問題と考えている。

ユーティリティについては、C言語で書かれた各種ソフトウェアを、インプリメントしてゆく予定であるが、外国製のものがほとんどで国内での販売代理店の認識がまだまだのところがあって、スムーズに進行したいのが現状であり、今後の各所の協力を期待するものである。また、

α-DOSをよりよいものにするために、自社内でもユーティリティの開発を進めているが、言語等の開発は、プログラム技術だけでなく、プログラムセンスのある人材が必要で、さらに費用、時間を含めて莫大な労力を要する。そのため、何を開発するかを決定することの方が大きな問題である。

9、おわりに

α-DOSの開発を振り返り、出来あがるにつれUNIXに似てしまったことを喜びにも思え残念にも思う。当社のような小さな会社でも、このような研究をしていることを知っていただければ幸いである。

謝辞.

システムの開発にあたって，オリジナルのC言語を提供いただいた，大阪大学基礎工学部情報工学科の都倉研究室の都倉信樹教授，ならびに都倉研究室のみなさまに深く感謝します。

またC言語のインプリメントを始め，種々のご指導ご協力をいただいた都倉研究室の辻野嘉宏，竹村治雄両氏に深く感謝します。

富士通万M-IIのハードウェアについて，各種の資料提供を心良くお引受けいただいた，富士通(株)，間様に深く感謝します。

また，技術的にご指導いただきました，森様をはじめとする飛田様，杉井様，その他技術の方々にも深く感謝します。

最後に，α-DOSの開発にたずさわった当社の金元，中島，増井，をはじめとする技術のみなさまに感謝します。

参考文献

1) B、W、カーニハン 他；プログラミング言語C，UNIX流プログラム書法と作法 共立出版

2) 栗原隆他：マイコンUNIXの使い方　産報出版

3) Richard Gaithier: Using the UNIX System　標準UNIXハンドブック　アスキー出版

4) 藤井純 他；オペレーティングシステム コンピュータサイエンスシリーズ 産業図書

5) 林 努 他：MC68000用OSの基本設計，ファイルシステムの設計
情報処理学会マイクロコンピュータ研究資料25 1982.12.2

6) Motorola：MC68000 Resident Structured Assembler Reference Manual