プログラミング環境 第2回

久野 靖 *

1990.10.8

4 ファイルシステムとデータ記憶

4.1 磁気ディスク装置、ファイルシステム

計算機というのは「情報を処理する」、もっと平べったく言うと「bit 列を加工する」ような装置だ、というのは前回にやった。ところで、情報を処理するためには蓄えておくことも当然必要になる (Q. なぜか?)。蓄えておく場所としては

・CPU の中、つまりどっかのレジスタ
・主記憶 (メモリ) の中

というのが当然考えられると思うが、それだけでは容量が足りないし、電源を切ると消えてしまう、という欠点がある。[1]

そこで容量が大きくて記憶が安定した、**2 次記憶装置**を接続して用いる。(Q. あなたはどんな 2 次記憶装置を知っているか?) その中でも、磁気ディスク装置が価格といい容量といい速度といいちょうど手頃なので、広く使われている。その概要を図 1 に示す。なおフロッピーディスクはこれの親戚だが円盤がぺらぺらで 1 枚だけケースに入っているやつである。

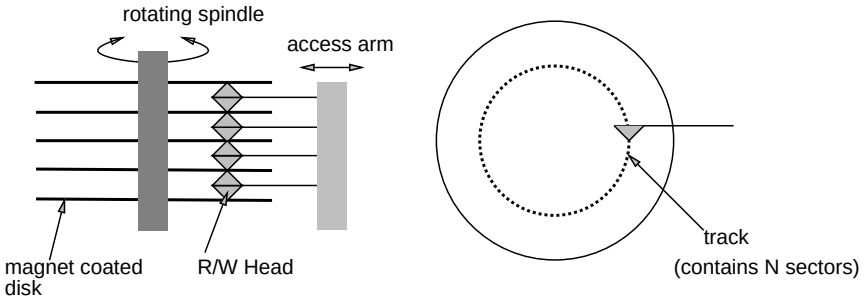

図 1: 磁気ディスク装置の概要

ところで、これにデータを読み書きするには、「どのトラックのどのセクタを読め/書け」と CPU が言えばいい。さて、そういうのが裸で計算機とつながっていたら、あなたは幸せか?(間) もちろん、幸せでないから OS が手助けしてくれるのだが。OS の中でも「ファイルシステム」とか「データ管理」とか呼ばれる部分はおおむね次のようなことをやってくれる。

*筑波大学経営システム科学専攻
[1]実はうそで、私が始めて使ったミニコンは磁気コアメモリだったから「内容を保存したまま電源を切る」ことができた。

・「どれだけの領域が欲しい」「返却する」という要求に対処。
・他の人と領域がかちあわないように管理。
・他の人のデータを壊したり、他人に勝手に読まれないよう管理。
・トラック番号、セクタ番号でなく名前で指定できるようにする。
・名前も他人とぶつかったりしないような機構を用意する。
・実際にデータを読み書きする手助け。

これは見かたによれば OS が「ディスク上の領域」という「資源」を管理していることになるわけである。

4.2 ファイル

一方、利用者はディスク上に置かれるデータの集まりを「ファイル」という概念で捉える。ファイルとはどんなものか?

・情報を蓄えておく場所/容れ物
・長期的、恒久的に
・付随する情報といっしょに

ファイルシステムより上のレベルから見ると、単なるセクタの並びでしかないはずのディスク装置が「ファイルの集まり」であるかのように見える。これはちょうど 1 個しかない CPU とのっぺらぼうの主記憶の上に「プロセス」の集まりがあるかのように見えるのと同様である。プロセスもファイルも現実には存在しないが OS によって「あたかもあるかのように」扱うことができる、計算機固有の「概念」である。以下では、ファイルの様々な側面について見てみよう。ここで取り上げるのは Unix の場合だけれど、多くの OS で通用する場合が多い。Unix 固有の部分も、それなりに面白い。

4.3 ファイルの名前

ファイルには、名前がついている (名前の長さ?使える文字?)。ファイルの名前を調べるには、`ls` 指令を使う。通常 `ls` は . で始まるファイル名は表示しない。なぜかというと、各種のプログラムごとに固有のオプション設定などを . で始まるファイルに書く、という習慣があって、それらのファイルがいつも表示されているとうるさいからである。`-a` をつければこれらもすべて表示される。

```
ls      -- 今いるディレクトリ (後述) にあるファイルの一覧を表示
ls -a   -- 加えて、「.」で始まるファイルも表示
```

たとえば私がとある機械でこれらをやってみた例を示す:

```
% ls
bin     hello.p work
% ls -a
.               .history        .newsrc         .xinitrc
..              .login          .rhosts         bin
.cshrc          .logout         .twmrc          hello.p
.emacs          .mailrc         .x11defaults    work
%
```

このように、実は結構沢山 . で始まるファイルがあるものである。
ところで、ファイルの名前にはどんな文字が使えると思うか?これまでの所、英数字は使えている。また上で見たように . も大丈夫そうである。(本当にそうか?... なんてのもありか?)*とか!とかはどうか?じつはこういう記号は特別な意味を持っていることがあるので厄介なのだが、’で囲んでやれば結構大丈夫。

```
% cp hello.p '*'
% ls
*       bin     hello.p work
% rm '*'
%
```

ついでに、ファイル名が何文字くらいまで可能かというのも興味深い問題である。

4.4 ファイルの中身

ファイルには、中身がある。(当たり前だと思いますか?)(中身のないファイルに利用価値はないか?) 中身とはは何か?「計算機は bit の列を加工する」という話だったから、ファイルも bit の列を格納する。しかし、普通は 8bit を 1 つのかたまり (バイト、と呼ぶ) と考え、バイトの列を格納することが多い (なぜか?)。Unix もそうである。その中身をバイトの列として調べるには (ダンプを表示する、とかいう):

```
od -c ファイル名    -- 文字形式で表示
od -x ファイル名    -- 16 進形式で表示
```

ファイルの中でも、文字ばかりを格納してあるファイルを**テキストファイル**と呼ぶ。テキストファイルがなぜ重要かというと、人間が直接読んだり書いたりできるのは (狭い意味では) テキストファイルだけだからである。これと対比してテキストファイルでないファイルを**バイナリファイル**と呼ぶこともある。テキストファイルを書いたり修正するのに**テキストエディタ**を使う。nemacs がその例である。ところで、ファイルに格納されるのはビットの列だったはず。どうやって文字が格納できるのだろうか?(間)

答えは、いくつか決まった長さのビットごとに、そのビットがどういうパターンだったらどんな文字、という約束というか対応関係をつけておく、ということである (符号化)。英数字記号に対しては ASCII と呼ばれる符号化が広く使われている (が、IBM 文明などでは EBCDIC というのもある。) これらでは 1 文字を 8 ビット (=バイト) に対応させている。Unix は ASCII 文明である。下に ASCII のコード表を示す。

00 NUL	01 SOH	02 STX	03 ETX	04 EOT	05 ENQ	06 ACK	07 BEL
08 BS	09 HT	0A NL	0B VT	0C NP	0D CR	0E SO	0F SI
10 DLE	11 DC1	12 DC2	13 DC3	14 DC4	15 NAK	16 SYN	17 ETB
18 CAN	19 EM	1A SUB	1B ESC	1C FS	1D GS	1E RS	1F US
20 SP	21 !	22 "	23 #	24 $	25 %	26 &	27 ’
28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 ‘	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C |	7D }	7E ~	7F DEL

これがあればビットの列からそれが何の文字かを知ることができる。次のファイルは何と書かれているか?

```
% od -x hello.p
0000000  7072 6f67 7261 6d20 6865 6c6c 6f28 6f75
0000020  7470 7574 293b 0a62 6567 696e 2077 7269
0000040  7465 6c6e 2827 4865 6c6c 6f2e 2729 2065
0000060  6e64 2e0a
0000064
%
```

もちろん、コード表で解読しまくってもいいけど...

```
% od -c hello.p
0000000    p   r   o   g   r   a   m       h   e   l   l   o   (   o   u
0000020    t   p   u   t   )   ;  \n   b   e   g   i   n       w   r   i
0000040    t   e   l   n   (   '   H   e   l   l   o   .   '   )       e
0000060    n   d   .  \n
0000064
%
```

4.5 ファイルの長さ

ファイルには、長さがある。(当たり前だと思いますか?)(長さの上限はいくつだと思いますか?。試さないこと。) 長さの単位はバイト。これを調べるには

```
ls -l   -- 長さを始めとする詳しい情報を表示。
```

によればよい。ところで、実は上の hello.p というファイルの中身は:

```
% cat hello.p
program hello(output);
begin writeln('Hello.') end.
%
```

さて、このファイルの長さはいくつだと思うか?なぜそうか?

```
% ls -l
total 3
drwxr-xr-x  2 kuno          512 Oct  3 16:05 bin
-rw-r--r--  1 kuno           52 Oct  3 16:06 hello.p
drwxr-xr-x  2 kuno          512 Oct  3 16:05 work
%
```

4.6 ファイルの種類

使う人にとっては、ファイルにはいろんな種類がある。

・テキストファイル (お手紙、プログラム、データ、...)
・実行形式ファイル
・オブジェクトファイル
・ライブラリファイル

しかし、「中身」の所に書いたように、Unix にとってはどれも同じ「バイトの列」で、互いに区別はない。ではどうやって区別するかというと。

・名前で区別する。(.p → Pascal プログラム、.c → C プログラム...)
・中身で区別する。(実行形式などは先頭に種別が入っている)
・自分で覚えておく。

例えばコンパイルした結果の実行ファイルなどだと:

```
% pc hello.p
% a.out
Hello.
% od -x a.out
0000000  8103 010b 0000 2000 0000 2000 0000 0000  ←始めの方が
0000020  0000 02e8 0000 2020 0000 0000 0000 0000    マジックナンバ
0000040  bc10 2000 d003 a040 9203 a044 952a 2002
```

```
0000060  9402 a004 9402 400a 1700 0010 d422 e148
0000100  0300 0008 c200 6200 8090 0001 0280 0004
0000120  0100 0000 4000 0008 0100 0000 4000 00a9
...
%
```

まあ、結局どこからか先は「自分で覚えておく」ことになるわけですよね。ところで、ファイルの種類を調べる指令もある。

```
file ファイル名...   -- ファイルの種類を調べる
```

ただし、これも名前や中身から「想像」するだけだからすぐ間違う。

```
% file *
a.out:      sparc demand paged dynamically linked executable not stripped
bin:        directory
hello.p:    ascii text
work:       directory
%
```

4.7 ファイルに付随する日付

`ls -l` ではそのファイルを最後に変更した日時も表示されるから、その情報もあることは分かる。ほかに、最後に読み出した日時も記録されている。

```
ls -t   -- 表示を変更日時の新しい順におこなう。
ls -lu  -- -l と同じだが、変更日時の変わりに読み出し日時を使う
```

なお、lsではこれらを自由に組み合わせていい。たとえば-a と-l と-t をまとめて指定して `ls -lat` というのも可能。ついでに、ファイル名を指定すればそのファイルに関する情報のみが表示される。

4.8 ファイルの持ち主、モード

`ls -l` ではそのファイルの持ち主 (作った人) も表示される。なぜ、持ち主の情報が必要か?(間) Unix では、ファイルの保護を

```
user(持ち主)    \      / read(読める)       \
group(グループ) -ごとに- write(書ける)      -をそれぞれ設定
other(その他の人 /      \ execute(実行できる)/
```

することにより行なう。このモードは `ls -l` の表示に含まれている。モードを設定するのは次による:

```
chmod [u][g][o](+|-)[r][w][x] ファイル...   -- モード設定
```

普段あなたが作ったファイルは「誰にでも読めて、自分のみ書ける」はずである。(それくらいがリーズナブルですよねえ?)

```
% cp hello.p zzz
% ls -l
total 28
-rwxr-xr-x  1 kuno        24576 Oct  3 16:42 a.out
drwxr-xr-x  2 kuno          512 Oct  3 16:05 bin
-rw-r--r--  1 kuno           52 Oct  3 16:06 hello.p
drwxr-xr-x  2 kuno          512 Oct  3 16:05 work
-rw-r--r--  1 kuno           52 Oct  3 16:50 zzz
% chmod ugo-rwx zzz
% ls -l zzz
----------  1 kuno           52 Oct  3 16:50 zzz
% cat zzz
```

```
zzz: Permission denied
% chmod ugo+r zzz
% ls -l zzz
-r--r--r--  1 kuno           52 Oct  3 16:50 zzz
% cat zzz
program hello(output);
begin writeln('Hello.') end.
% cp hello.p zzz
cp: zzz: Permission denied
% chmod u+w zzz
% ls -l zzz
-rw-r--r--  1 kuno           52 Oct  3 16:50 zzz
% cp hello.p zzz
%
```

4.9 ファイルの i-番号

i-番号というのはファイル一つ一つにつく固有番号である (PID のようなものと思えば良い)。`ls -i` でこれを表示させることができる。なぜ名前があるのにそんなものが必要なのか?実は、名前は変化することがある。また、一つのファイルに別の名前をつけることもできる。

```
mv ファイル名 新しい名前    -- 名前を変更する
ln ファイル名 新しい名前    -- ファイルに新しい名前をつける
rm ファイル名               -- 名前を無効にする
```

実は、rm はファイルを消すのではなかった!ただ、全ての名前が無効になったファイルはそれ以上触りようがなくなるので結果として消えるわけである。

```
ls -i
 37806 bin        7282 hello.p    7281 work       7283 zzz
% mv zzz xxx
% ls -i
 37806 bin        7282 hello.p    7281 work       7283 xxx
% ln xxx yyy
% ls -i
 37806 bin        7282 hello.p    7281 work       7283 xxx        7283 yyy
% ls -li xxx yyy
  7283 -rw-r--r--  2 kuno           52 Oct  3 17:06 xxx
  7283 -rw-r--r--  2 kuno           52 Oct  3 17:06 yyy
% chmod u-w xxx
% ls -li xxx yyy
  7283 -r--r--r--  2 kuno           52 Oct  3 17:06 xxx
  7283 -r--r--r--  2 kuno           52 Oct  3 17:06 yyy
%
```

ところで、`ln` の類似品にこんなのもある。

```
ln -s 名前1 名前2    --- 名前2なるシンボリックリンクを作る
```

シンボリックリンクというのは、名前 2 が現れたらそれは名前 1 のことだと思う (置き換える) という機能で、ただの `ln`(こちらはハードリンク、などと呼ばれる) によく似てはいる。

```
% ln -s xxx zzz
% ls -li
total 5
 37806 drwxr-xr-x  2 kuno          512 Oct  3 16:05 bin
  7282 -rw-r--r--  1 kuno           52 Oct  3 16:06 hello.p
  7281 drwxr-xr-x  2 kuno          512 Oct  3 16:05 work
  7283 -rw-r--r--  1 kuno           52 Oct  3 17:06 xxx
  7284 lrwxrwxrwx  1 kuno            3 Oct  3 17:15 zzz -> xxx
```

```
% cat zzz
program hello(output);
begin writeln('Hello.') end.
% echo Hi. >>xxx
% cat zzz
program hello(output);
begin writeln('Hello.') end.
Hi.
```

4.10 その他の良く使う指令

ここまでに出てこなかったがファイルに関連してよく使う指令のリストを掲げておく。

```
cp ファイル1 ファイル2        -- ファイルのコピー
egrep パターン ファイル...    -- パターン探索
wc ファイル...                -- 字数、語数、行数を数える
cat ファイル...               -- ファイルの中身を画面に表示
less ファイル...              -- 1画面ずつ止まりながら表示
head ファイル                 -- ファイルの先頭部分のみ表示
tail ファイル                 -- ファイルの終り部分のみ表示
diff ファイル1 ファイル2      -- 2つのファイルの違いを表示
```

4.11 おまけ:漢字コードについて

ASCII というか、英数字記号は 8 ビットでうまいこと入るのは前述の通りである。しかし、問題は漢字である。8 ビットに入り切る文字の数は 256 種類しかないので、漢字には全然足りない。そこで漢字を扱うには 2 バイトを 1 文字として符号化をする。符号化のしかたは、JIS 規格に従う。ところで、ASCII の文字と JIS 漢字とでは重なる部分があるので (例えば「!"」は何と「、」と同じである)、これらを区別して扱う方法が必要である。我々のところでは、普通は ISO で規定されている 3 バイトの列で「ここから漢字」「ここから ASCII」を指定している。

```
% cat t.txt
1. ファイルの一覧を
  表示するには ls を使う。
% od -x t.txt
0000000  312e 201b 2442 2555 2521 2524 256b 244e
0000020  306c 4d77 2472 1b28 4a0a 2020 1b24 4249
0000040  3d3c 2824 3924 6b24 4b24 4f1b 284a 6c73
0000060  1b24 4224 723b 4824 2621 231b 284a
0000076
```

つまり ESC-$-B が「ここから漢字」、ESC-(-J が「ここから ASCII」になっている。これが一応由緒正しい JIS に従った符号化である。

ところが、漢字に出入りするたび 3 バイト費やすのはいやだ、という意見もある。そこで、ASCII では 8 ビットのうち頭の 1 ビットは常に 0 であることを利用し、漢字は JIS コードを頭のビットのみ 1 に変更したもので表せば両者を区別できる。これを EUC と呼んでいる。

```
% nkf -e t.txt >t.euc
% od -x t.euc
0000000  312e 20a5 d5a5 a1a5 a4a5 eba4 ceb0 eccd
0000020  f7a4 f20a 2020 c9bd bca8 a4b9 a4eb a4cb
0000040  a4cf 6c73 a4f2 bbc8 a4a6 a1a3
0000054
```

我々のところでは実はこれも混用できる。由緒正しい JIS では 7 ビット目は 0 なので、EUC と混ざっていても混乱することはないから大丈夫である。

さらに困ったことに、パソコンの世界ではシフト JIS なるものが流布している。これは漢字は 1 バイト目のみ頭のビットが立ち、2 バイト目は ASCII の記号の部分をよけるように符号化したものである。

```
% nkf -s t.txt >t.sj
% od -x t.sj
0000000  312e 2083 7483 4083 4383 8b82 cc88 ea97
0000020  9782 f00a 2020 955c 8ea6 82b7 82e9 82c9
0000040  82cd 6c73 82f0 8e67 82a4 8142
0000054
```

このように、漢字を含むテキストの場合 3 種類の符号化がいり乱れているので注意が必要である。とりあえず情報科学科の Unix では JIS が標準 (たとえば nemacs で普通に書き出すと JIS) だが、NEWS と X 端末では EUC のファイルもそのまま表示できる。一方、PC から持ってきたものは SJIS のままであるのが普通なので自分でコード変換しないといけない。ちなみに、コード変換指令は次の通りである。

```
nkf ファイル >出力ファイル       -- JIS に変換
nkf -e ファイル >出力ファイル    -- EUC に変換
nkf -s ファイル >出力ファイル    -- SJIS に変換
```

また NEWS の file 指令では「漢字コードがどの体系か」も教えてくれるので試してみられたい。

A 演習および課題

A.1 4 節の演習

4-1. ファイル名として、どんな字が使用できるか、実地に調べてみよ。[2]

4-2. 同様に、ファイル名の長さはいくつくらいまで可能か、実地に調べてみよ。[3]

4-3. 友人とあんまり長くない (100 バイトくらいまでがいいと思う) テキストファイルの 16 進ダンプのプリントアウトを交換し、どちらが先に互いのファイルの中身を正確に言い当てるか競って見よ。

4-4. ファイルのモードをいろいろ変更し、ls で正しく変更されているか確認せよ。特に「自分に読めない」「自分に書けない」などの設定が確かにそのように働いているか確認してみよ。また、「誰にでも読めるが、自分には読めない」などの変な設定が意味を持つかどうか検討してみよ。

4-5. mv で名前を変更するのと cp で複写してもとのを消すのではどう違うか、実際に違う点を示せ。また、ln で別の名前をつけてから rm でもとの名前を消すのではどうか?

4-6. ln と ln -s のどちらもあるファイルに別の名前をつける、という使い方をすることができる。ではこれらで違いが出るのはどのような場合か?実地に試して報告せよ。[4]

A.2 2 回目の課題

本日の課題から報告を提出する場合は、上記 4-1〜4-6 のうちから最低 2 つ以上選択して下さい。

---

[2]既に説明したように、" で囲まないと使えない字があるが、それは当然囲めばいいわけ。囲んでもだめな字はあるか? 手当たり次第にやるのではなく、系統的にやって欲しい。

[3]これを aaaa... とかいって数を数えながらファイル名を打ち込んでやる、というのはあんましかっこよくない。次のような方法を試してみよ。

```
set a=0123456789
cp hello.p $a
```

こうすると、$a というのは 0123456789 のことだから、長さ 10 文字のファイルが作ってみられたわけ。じゃあaaaa$a は...? などのようにする。

[4]ヒント:別名をつけたあとで、もとのファイルを mv したり rm したりしたらどうなるかな?

B　前回の資料の訂正

3 節の図 1 が間違っていました。すいません。次の図が正しいので、差し替え — られないでしょうけど、一応掲載しときます。

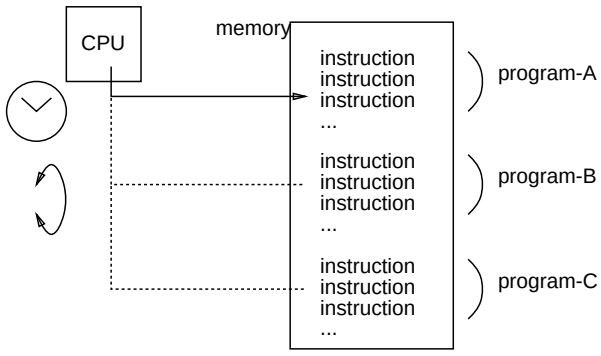

図 2: 正しい図 3-1