本稿は Linux Japan 誌 1999 年 5 月号に掲載された記事に補筆修正したものです.

仮題：時計 その 2

桜前線が日本を北上する季節になりました．卒業生を送り新入生を迎えるこの時期には，やっぱり一年間何もせずに過ごしてしまったと反省し，今年こそは教育に研究に力を注ごうと決意を新たにするのですが... 花見が，団子（私あまり酒が飲めないので）がどうもいけません．前回に引続き,『時間』にかかわるお話しに暫くおつき合いください.

皆さんの X のデスクトップには，多分『時計』が立ち上がっていると思います．余計なアプリケーションは立ち上げず，Mule 一本で過ごすという方も，ステイタスラインに時刻くらいは表示させている（.emacs に (display-time) を記述します）ことと思います．すなわち，常に時刻が表示されているのですが，その値は正しいでしょうか．腕時計や時報と照らし合わせたことがありますか．もし，違っていた場合どうしたらいいのでしょうか.

システムクロックの表示・設定：date

時刻を扱うコマンドで最も大切なのは date コマンドでしょう．単に date とすれば現在時刻を表示します.

```
date ↵
Sun Feb 7 21:27:46 JST 1999
```

date はいろいろな書式で日付や時刻を表示することが可能です．書式を変更するには オプション “+” を使います．例えば,

```
date +'%s %c' ↵
918783892 Fri Feb 12 10:44:52 1999
```

は，%s：1970 年 1 月 1 日 0 時 からの経過秒数（もっとも基本的な数値），%c：経過秒数を日付と時刻に変換して，時差を加えて表示する指定となります．ここで時差とは，海外旅行でおなじみの協定世界時（英国のグリニッジ標準時）に対する時差です．日本は，+9 時間進んでいますね．逆に言うと，グリニッジ標準時は 9 時間を引いた値となります．これを表示させるには，オプション -u を指定します.

```
date +'%s %c' -u ↵
918751492 Fri Feb 12 01:44:52 1999
```

確かに 9 時間遅れて表示されました.

date による時刻合わせ

時刻を合わせるということは，システムクロックの値を変更することです．誰でも変更できてしまことは望ましくはないので，root 権限でのみ可能です．MMDDhhmm[CC[YY][.ss]] の書式で日付・時刻（時分）・年・秒を設定します．例えば，以下のようになります.

```
date 021211121999.15 ↵
Fri Feb 12 11:12:15 JST 1999
```

最近の date はオプション -s が追加され，色々な書式を受け付けるようになりました.

```
date -s "Feb 12 11:12:15 1999" ↵
Fri Feb 12 11:12:15 JST 1999
```

ここで，時刻を大幅に変更すると X が落ちたりしますから要注意です．date コマンドは，現在時刻の表示以外にも，日付の計算などもできますから，マニュアルを一度見ると面白いです．日本語マニュアルに感謝 m(_ _)m.

date を使った時計

プログラムという程でもないかもしれませんが，date を使って日付と時刻を表示するシェルスクリプトをこしらえて遊んでみましょう．概略は次のようなものです．全体を while 無限ループに入れています (^^;．それだけでは負荷が高くなるので，sleep コマンドを用いて間隔を 1 秒間空けています．また，date コマンドで改行してしまわないように echo -n 等を使ってみました．簡単に日本語が使えるので，ちょっと嬉しいかもしれません.

date.sh

```
#!/bin/sh
while { sleep 1 } do
  wnum=`date +"%w"`
  case "$wnum" in
    0) week="日" ;;
    1) week="月" ;;
    2) week="火" ;;
    3) week="水" ;;
    4) week="木" ;;
    5) week="金" ;;
    6) week="土" ;;
  esac
  echo -ne '\r';
  echo -n `date +"%H 時%M 分%S 秒 %m 月%d 日"`
  echo -n "($week)"
done
```

背景に pixmap 画像が貼り付けられる漢字端末（rxvt,aterm）等で実行させると良いでしょう．例えば,

```
rxvt -geometry 26x1 +sb -fn a24 -fk k24 \
     -name rxvtclock -fg gold \
     -pixmap 'bw.xpm' -e date.sh
```

のような具合です．-fn a24 で英数フォント -fk k24 で漢字フォントを指定しています．やや大きめの 24 ピクセルのものを指定しました．+sb はスクロールバーの表示を止めるオプションです．-name rxvtclock としてわざわざ名前を付けたのは，ウィンドウマネージャがタイトルバーを付けるのを禁止する目的です．もちろん，例えば afterstep なら .steprc で

```
Style "*clock"  Notitle, ***
```

と設定されていることを前提に考えています．

背景の pixmap 画像 “bw.xpm” は標準で適当なものがなかったので，自分で作成しました．例えば，グラデーションがかかった背景模様を作成するプログラム **bggen** を用いて

```
bggen -h 28 -w 56 gray80 black > bw.ppm
convert bw.ppm  bw.xpm
```

などとして，作ってみてください．この連載で紹介した **xart** なんかもこの位の小さいビットマップを作成するには好都合なツールです．

図 1　date を利用した日本語時計端末

時刻合わせ

root 権限があれば date によりシステムクロックを変更できます．次に問題となるのは，本当に正確な時刻をどこから得るかということです．Unix の世界では NTP(Network Time Protocol) という仕組みが用意されていてネットワークに繋がれたホストの時刻を同期させることができます．しかし，これは基本的に常時接続されていることが前提なので，ダイアルアップで不定期に接続するホストでは使いにくいようです．また NTP では同期すなわちシステムクロックの進み具合を合わせる目的があるのでやや大袈裟となります．一日数秒程度しかシステムクロックが狂わないならば，とにかく一回時刻を合わせればよいと，気楽に構えましょう．すると，色々な方法が考えられます．例えば，時報に合わせるとか，時報に合わせておいた腕時計に合わせるとかです．もう少しスマートな方法もあります．

netdate，radate，nist

ネットワークに接続された信頼できるホストと時刻を合わせるためのコマンドがあります．rdate, netdate, nist などが有名です．少しずつオプション指定が違いますが基本的には，

```
command hostname
```

のように，ネットワーク上にあるホストを指定して信頼できる（と思われる）時刻を取得します．root 権限で実行すると，得られた値をシステムクロックに設定できます．ですから，ダイアルアップユーザーも，プロバイダーのメールサーバーなどは多分正確な時刻を保持していると考えて，そのホストに合わせれば良いことになります．常時接続ホストならば，毎夜 cron で実行させるのもいいでしょう．では，netdate を試してみましょう．自分自身=localhost に時刻を尋ねてみます．

```
/usr/sbin/netdate localhost ↵
Local host aya has best time, so not setting
date
localhost +0.000       Mon Feb 15 00:17:32.423
```

一致するのが当り前なので，無意味ですかね．ネットワーク接続された場合には

```
/usr/sbin/netdate cckmail ↵
cckmail -0.091         Mon Feb 15 14:46:25.000
```

時間のずれも表示され設定が行われます．ずれが大きい場合に，実際に設定するかどうかも指定できます．

CMOS クロックの操作：/sbin/clock

UNIX 系 OS の常識ではホストの電源は常時 ON で，システムクロックが時間を刻んでくれています．が，省電力のために，あるいはノートユーザーでは必然的に，電源 OFF 状態から頻繁に再度起動する方も多いでしょう．その際には，時刻はコンピュータ内蔵の CMOS クロックチップから読まれます．ところで，この CMOS クロックの時刻は正常でしょうか？ 昔，起動して暫くしてからファイルの更新日時が全く違っているのに気付き，大慌てしたことがありました．BIOS 設定で時刻を合わせておかなかったのが原因です．そういう訳で，CMOS クロックも一度は時刻合わせをした方が良いのでしょう．もちろん，その目的に合わせたツールが存在します．ずばり “clock” です．ところで，あまりに一般的な名前なので，わざわざ “/sbin/clock” と書かねばなりません．なぜなら他にも “clock” という同名の時計がインストールされている場合があるからです．さて，本題に戻って root 権限で clock を実行す

ると，コンピュータに内蔵されている CMOS クロックの読み書きができます（一般ユーザーではたぶん拒否されます）.

```
/sbin/clock -r ↵
Fri Feb 12 10:32:08 1999
```

CMOS クロックをグリニッジ標準時とみなして，タイムゾーンの補正を加えて表示する場合は，オプション -u を用います.

```
/sbin/clock -ru ↵
Fri Feb 12 19:32:08 1999
```

CMOS が既に日本時間になっていれば，時差は +9 時間ですから，このように +9 時間進んで表示されます.

読み込むと同時にシステムクロック値を変更するには，オプション-s を指定します．逆に，システムクロック値を書き込むにはオプション -w を指定します．ここで，ちょっと悩みがあります．それは，CMOS クロックにはグリニッジ標準時刻と現地時刻のどちら保持するかを決めなければならないことです．どちらでも構わないのとは思いますが，CMOS クロック自身にはどちらで保持しているという情報がないので標準時を保持する場合には注意が必要です．すなわち，読む場合も書く場合も CMOS クロックを標準時とみなすオプション - u を一緒に付けなければなりません．お使いのシステムがどちらを採用しているかは起動時に実行される /etc/rc.d 以下のスクリプトの中身をみればわかります．Slackware では

```
grep clock /etc/rc.d/* ↵
/etc/rc.d/rc.S:# Configure the system clock.
/etc/rc.d/rc.S:if [ -x /sbin/clock ]; then
/etc/rc.d/rc.S: /sbin/clock -s
```

となっており，現地時刻で統一している様です.

hwclock

clock よりも多機能な CMOS クロックツール hwclock がインストールされているかもしれません．マニュアルを読むと，CMOS クロックとシステムクロックについての説明があります．また，ここで触れませんでしたが，時計の進み具合を調節する **adjtimex** との関連も説明されており，特にノートユーザーの方は一読の価値があると思います．デスクトップユーザーにとっての恩恵は，オプション --test でしょう．その名の通り『お試し』で，実際の書き込みが行われませんから安心してコマンドをいじくれます．そのうち，/sbin/clock は/sbin/hwclock のリンクになっているかも...

C の標準関数

以上，時間を扱うコマンドを紹介しました．これらのコマンドもシステムクロック値を取得する場合，標準関数やデータ構造を使っている筈です．どんなものがあるかは /usr/include/time.h に書いてありますから，そのうち，もっとも基本的な関数や構造体を調べて使ってみましょう（あと，“man -k time” 等で関連事項を列挙することもお忘れなく）.

time()

もっとも基本的な数値である，1970 年 1 月 1 日 0 時からの経過秒数を返す関数です．値を格納する変数を（ここでは now ） time_t 型で宣言して

```
time_t now;
...
time(&now);
```

とします．ポインタで渡すのが嫌で，time() の戻り値を代入して使いたいならば

```
time_t now;
...
now = time(NULL);
```

と time() には NULL を渡す方法もあります.

tm 構造体

日付（年・月・日・曜日）や時刻（時・分・秒）などを要素に持つ構造体が time.h で定義されていますね．注釈がないので， man localtime を実行しましょう．すると，日本語マニュアルがインストールされていれば,

```
struct tm
{
  int  tm_sec;     /* 秒 */
  int  tm_min;     /* 分 */
  int  tm_hour;    /* 時間 */
  int  tm_mday;    /* 日 */
  int  tm_mon;     /* 月 */
  int  tm_year;    /* 年 */
  int  tm_wday;    /* 曜日 */
  int  tm_yday;    /* 年内通算日 */
  int  tm_isdst;   /* 夏時間 */
};
```

という情報が途中に表示されます．大変便利そうな構造体ですが，要素を確定しなければ意味がありません．それには，マニュアルの最初の部分にある，書式

```
struct tm *localtime(const time_t *timep);
```

に従って，localtime( ) に経過秒数を値とする time_t

型の変数のポインタを与えれば良いことが判ります．他にもいろいろありそうですが,『案ずるより生むが易し』で，日付と時刻を表示するプログラムを作ってしまいましょう．

```
#include <stdio.h>
#include <time.h>

char *week[]
      ={"日","月","火","水","木","金","土"};

void main()
{
  struct tm *now;
  time_t now_t;

  while (1){
    time(&now_t);
    now = localtime(&now_t);

    printf("%2d 時%2d 分%2d 秒 ", \
      now->tm_hour, now->tm_min, now->tm_sec);
    printf("%2d 月%2d 日 (%s)\r", \
      now->tm_mon, now->tm_mday, \
                  week[now->tm_wday]);
    fflush(stdout);
    usleep(300000L);
  }
}
```

大変簡単ですから説明は不要でしょうが，少しだけ注釈を．`fflush(stdout)` はバッファを吐き出すための常套手段です．無限ループが負荷を高めないように，`usleep( )` で空きを入れています．引数の単位は μs ですから，300000L では 0.3 秒間隔のつもりです．このままでは date.sh と同じ機能しかありませんし，行数も長い，コンパイルもせねばならないと負けてます．スクリプトで実現できる場合はどうしてもこうなりますね．X のプログラムで C に頑張ってもらいましょう．

簡単な X 上の時計

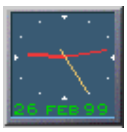

図 2　afterstep のアナログ時計アプレット：astime

X 上の時計をプログラムしようと考えた場合，画面更新を while ループ内に埋め込んでしまうと決めれば，時刻の取得に関しては，上述の C のルーチンがそのまま流用できます．したがって，あとは文字で表現するか (デジタル)，針で示すか（アナログ）の選択になります．ちょっとその前に，前回紹介し忘れたアナログ時計があります．afterstep 用のアナログ時計 astime[1] です（図 2）．日付が下の方に表示されているところが特徴ですね．astime の main ルーチンを示しますと，

```
void main(int argc, char** argv)
{
  defaults();
  parsecmdline(argc, argv);
  initialize(引数略);
  while (1) {
    update();
    usleep(X11_INTERVAL);
  }
}
```

などとなっておりまして，まあ見事に while ループで，負荷が高くならないように適当にインターバル（X11_INTERVAL）が入っているという構造です．もちろん，update( ) という手続きの中身が問題なのですが，ご自身で覗いてみてください，これもまた簡素極まりないものです．大きさ固定で，しかも頻繁に書き換え（再描画）を行っているので，本来面倒とされる，他のウィンドウに隠れていた部分が露出されたときの再描画などを考えなくてよくなっています．ただ単に時刻に応じて画面を更新してゆくだけの，date.sh と同じ感覚で済ませることが可能です．

ここで，本当に簡単なアナログ時計を作ってみましょう．前回紹介した拙作 xawclock を簡約したものです．`drawClock()` が描画部分で，時間を取得した後，針の角度を計算して `XDrawLine(...)` で描きます．`XFillArcs(...)` は 15 行目で宣言され，77〜83 行で要素を確定された弧の構造体配列を描く関数です（すなわち，円形の文字盤マークをまとめて描いています）．一般に，弧を一つ描く関数 `XFillArc(...)` で個別に描くよりも全体の速度が向上するといわれています．98 行目 `XSetLineAttributes()` はグラフィックコンテキスト gc_hands の線属性を設定しています．幅を 2 に設定するのが主な目的です [2][W3].

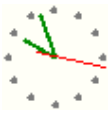

図 3　簡単アナログ時計：xsclock

図 3 に xsclock の様子を示します．実は，このプログラムはウィンドウマネージャへの登録（`XSetWMProperties( )` 関数を使う）を行っていないので，大変お行儀の悪いものとなってます．したがって，afterstep が Wharf ボタンに収めようにも，情報不足で収めることができません．XSetWMProperties( ) をきちんと書くにはあと 20 行程余分にかかります．そ

れを避けるには，初めから X ツールキット（Xt）を用いる方が得策ですので，機会があったらそちらで説明するということで，ここは目をつむってください．

xsclock.c

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <unistd.h>
 4: #include <math.h>
 5: #include <time.h>
 6: #include <X11/Xlib.h>
 7:
 8: #define setfg(x,y) \
 9:         XSetForeground(dpy,x,GetColor(y))
10: #define SIZE 56
11: struct tm *tmval;
12: char *color_bg="ivory";      /* 背景色 */
13: char *color_hands="green3"; /* 時分針の色 */
14: char *color_sec="red";       /* 秒針の色 */
15: char *color_num="gray50";    /* 文字盤マーク色 */
16: XArc Numbers[12];            /* 文字盤マーク位置 */
17:
18: Display *dpy;
19: Window root, win;
20: int scrn;
21: GC gc, gc_sec, gc_hands, gc_num;
22:
23: unsigned long GetColor(char *color)
24: {  /* 色の取得 */
25:   Colormap cmap;
26:   XColor c0, c1;
27:   cmap = DefaultColormap(dpy, 0);
28:   XAllocNamedColor(dpy, cmap, color, &c1, &c0);
29:   return(c1.pixel);
30: }
31:
32: void drawClock()
33: {
34:   int deg_sec, deg_min, deg_hour;
35:   time_t current_time;
36:
37:   current_time = time(NULL);
38:   tmval = localtime(&current_time);
39:
40:   deg_sec = tmval->tm_sec*6.0;
41:   deg_min = tmval->tm_min*6.0;
42:   deg_hour = tmval->tm_hour*30.0 + deg_min/12.0;
43:
44:   XFillRectangle(dpy,win,gc,0,0,SIZE,SIZE);
45:   XFillArcs(dpy, win, gc_num, Numbers, 12);
46:   XDrawLine(dpy, win, gc_hands, SIZE/2, SIZE/2,
47:             SIZE/2+18*sin(M_PI*deg_hour/180.0),
48:             SIZE/2-18*cos(M_PI*deg_hour/180.0));
49:   XDrawLine(dpy, win, gc_hands, SIZE/2, SIZE/2,
50:             SIZE/2+23*sin(M_PI*deg_min/180.0),
51:             SIZE/2-23*cos(M_PI*deg_min/180.0));
52:   XDrawLine(dpy, win, gc_sec,
53:             SIZE/2-10*sin(M_PI*deg_sec/180.0),
54:             SIZE/2+10*cos(M_PI*deg_sec/180.0),
55:             SIZE/2+27*sin(M_PI*deg_sec/180.0),
56:             SIZE/2-27*cos(M_PI*deg_sec/180.0));
57: }
58:
59: void main(int argc, char **argv)
60: {
61:   int i;
62:   for (i = 0; i < 12; i++) {
63:     Numbers[i].x = SIZE/2+25*cos(M_PI*i/6.0)-2;
64:     Numbers[i].y = SIZE/2+25*sin(M_PI*i/6.0)-2;
65:     Numbers[i].width = 4;
66:     Numbers[i].height = 4;
67:     Numbers[i].angle1 = 0;
68:     Numbers[i].angle2 = 64*360;
69:   }
70:
71:   dpy = XOpenDisplay(NULL);
72:   scrn=DefaultScreen(dpy);
73:   root=RootWindow(dpy,scrn);
74:
75:   win = XCreateSimpleWindow(dpy,root,0,0,
76:                             SIZE,SIZE,0,0,0);
77:   gc=XCreateGC(dpy,win,0,NULL);
78:   gc_sec=XCreateGC(dpy,win,0,NULL);
79:   gc_hands=XCreateGC(dpy,win,0,NULL);
80:   gc_num=XCreateGC(dpy,win,0,NULL);
81:   setfg(gc,color_bg);
82:   setfg(gc_sec,color_sec);
83:   setfg(gc_hands,color_hands);
84:   setfg(gc_num,color_num);
85:   XSetLineAttributes(dpy,gc_hands,2,LineSolid,
86:                      CapRound,JoinMiter);
87:   XMapWindow(dpy,win);
88:
89:   while(1) {    /* 時計を描く無限ループ (^^; */
90:    drawClock();
91:    XFlush(dpy);
92:    usleep(300000L);     /* ここは適当に調整 */
93:   }
94: }
```

0

xsclock.c を編集したら，

```
gcc -o xsclock xsclock.c -lX11 -lm ↵
```

のようにコンパイルして実行ファイル xsclock を作成してください．すぐにコンパイルできますから，色を変えたりして楽しんでください．

調子に乗って

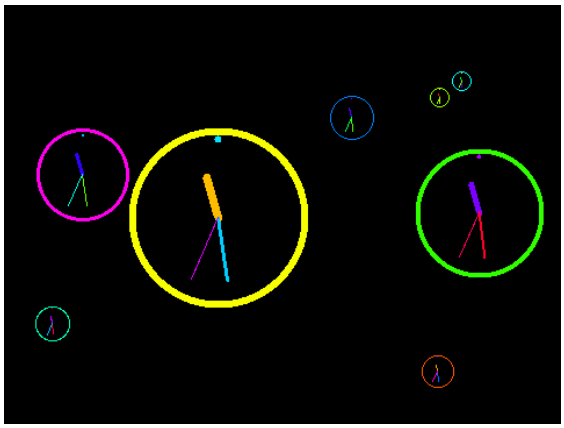

図 4　xlock の clock モード

X 上のスクリーンセーバー用のアニメーションを作る話を連載第 3 回目にしました．そのなかの，Linux のペンギンロゴが画面を徘徊する例を思い出してください．それにちょっと手を加えて，腹時計を追加してみましょう．窓に Pixmap を貼り付けた後，腹の部分に時計を描くだけのものです．xlock の clock モードでは，時計が画面に固定されたままで，作者の方には申しあげにくいのですがつまらないと思ってました．図 5 に

様子を示します．もちろん任意の pixmap を背景にして楽しむことができます [3]W3.

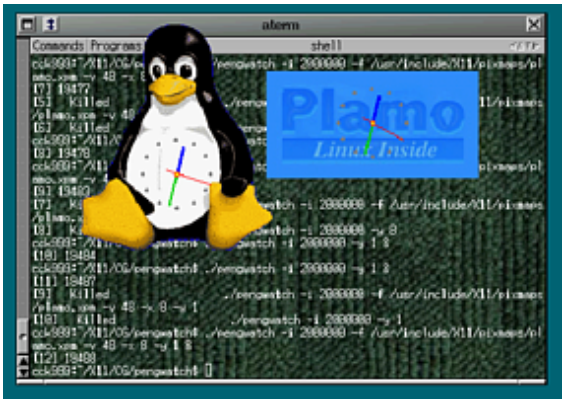

図 5　ペンギン腹時計：pengwatch

次週予告

いやープログラムの話はいつもひやひやもんです (^^；次回は，時計以外のデスクトップ小物の楽しい楽しい紹介と行きましょう．Xbiff の系譜やシステムモニターなど盛り沢山の予定です．

参考文献

[1] 下記 URL でまだミラーされています．
ftp://mirror.nucba.ac.jp/mirror/afterstep/apps/astime/ W3

[2] 筆者ホームページ 「X11 時計ギャラリー」をご覧ください．
http://ayapin.film.s.dendai.ac.jp/~matuda/Clocks/ W3

[3] X11 時計ギャラリーをご覧ください．
http://ayapin.film.s.dendai.ac.jp/~matuda/Clocks/clocks2big.html W3