HYAKU

MANGOKU

12

1998

百萬石

MMA

部長の言葉

ikidou@mma.club.uec.ac.jp

1998/11/20

本日は調布祭に御来場いただき、ありがとうございます。MMA は Microcomputer Making Association の略であり、コンピュータを製作するサークルとして発足しました。ハードに関する技術が発展した今において MMA ではソフト製作等、各自が興味を持っているものを自由に取扱っています。

百萬石は新入生勧誘時の春と、調布祭時の秋の年に二回の発行となっており、活動内容の発表等が書かれています。お楽しみください。

SSH 使い方講座

kyogoku@mma.club.uec.ac.jp

○はじめに…＜なんで ssh なの？＞

インターネットは元々学術目的で開発されたという経緯があるために、その通信方法は平文、つまり打った文字や表示される文字がそのままの状態でネットを行き来するのが基本となっています。

しかし、これでは悪意を持った人が通信を覗くと、流れている情報が全て筒抜けなのでメールの内容を読まれてしまったり、パスワードを盗まれてあなたのアカウントでログインされたりと言ったことが起こる可能性があります[1]。

仮にアカウントが乗っ取られてしまった場合には、被害は乗っ取られた本人に留まらず、サイト全体が脅威に晒されてしまいます。また乗っ取られたアカウントを踏台に更に別のホストを攻撃したりと言うことになれば、インターネット全体に迷惑をかけてしまうことになります。

そこでとりあえずパスワードが洩れない為の自衛策として、

1. パスワードを打たないでログイン出来るようにする。
2. 本物のパスワードを使わずに、毎回変わる様な使い捨てパスワードを使う。
3. 通信を暗号化してしまう。
4. 怖いので外からログインしないことにする:-P

と言った方法が考えられます。実際の方法にもよりますが、上から順にセキュリティーは高くなっていきます。

今回取り上げる ssh とは３番目の方法の１つで、通信を暗号化するためにパスワードは洩れない上に、メールを盗み見されてしまう心配もありません。

しかも一旦設定をしてしまえば簡単な手続きでログインできるのでお勧めです。

○ ssh について

unix を使っている人なら、rlogin や rsh、rcp 等のコマンドを知っていると思います。ssh とは、これらの「r」を「s」に置き換えて、slogin、ssh、scp といったコマンド名で、それぞれ「r」系のコマンドと同じ事が出来ます。簡単に説明すると、slogin はネットワーク越しにログイン、ssh はコマンドを実行、scp はファイルのコピーといった具合です[2]。

ただし、ssh を使えば何度も言ったようにその通信は全て暗号化される上に、X-プロトコルを暗号化して飛ばしたり、暗号化した TCP 的なトンネルを作って様々な通信をそこに通すといったことも出来ます。

ssh は暗号化の手順として、RSA 暗号というものを使っています。RSA 暗号とは他人に見られても構わない「公開鍵」と自分の手元にだけ置いておく「秘密鍵」という２つの鍵 (といってもランダムはアルファベットと数字の羅列ですが) を使って認証を行うものです。

○実際の使い方

さて、それでは実際に ssh を使ってみましょう。

---

[1] 特に ftp でのログインはパスワードを盗まれやすいので注意

[2] 実際には slogin と ssh の実体は同じ物ですけど

・Windows95を使ってリモートホストにsshでログインする
～Windows95/98でのSSH～

○ TeraTermProとTTSSH

これまでWindowsでsshを使うためにはかなり面倒な手順が必要で、しかも日本語を通してくれなかったり色々問題があったのですが、ここで紹介するTTSSHが登場してからは非常に手軽にsshを使えるようになりました。

TTSSHとはWindowsでの有名なターミナルソフト、TeraTermPro用のエクステンションでつまりTeraTermでsshが使えるようになるのです。

○具体的な手順

それではTTSSHを使ってみましょう。

まずはTeraTermProをインストールして下さい。TTSSHを使うためにはVer 2.3以上のTeraTermProが必要です。ダウンロードは以下の所で出来ます。

・Vector http://www.vector.co.jp/vpack/browse/software/win95/net/sn067018.html

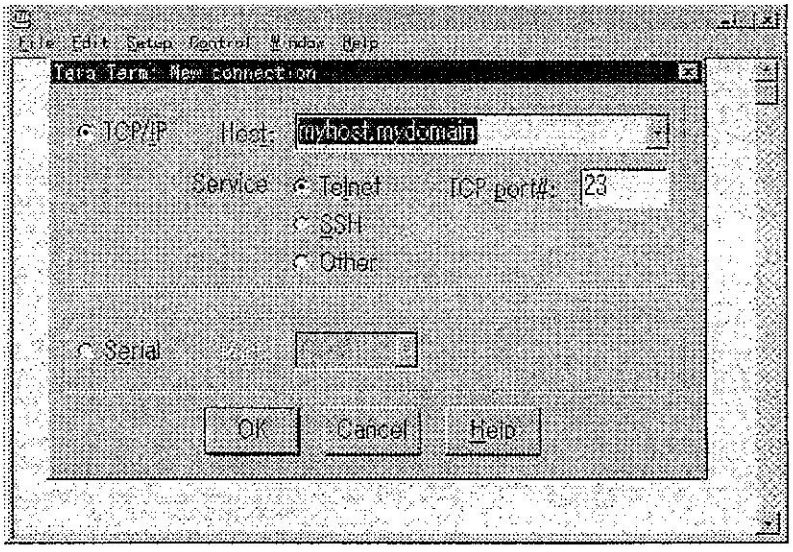
・窓の杜 http://www.forest.impress.co.jp/term.html#terat

次にTTSSHをインストールします。以下よりダウンロードして下さい。

ftp://ftp.htp.org/pub/crypto/ssh/ttssh/ttssh13.zip

ダウンロードしたアーカイブを解凍し、中身を全てTeraTermProがインストールされているフォルダにコピーして下さい。

これで準備完了です。先程コピーした、ttssh.exeをダブルクリックして見て下さい。以下の様な画面が出れば成功です。

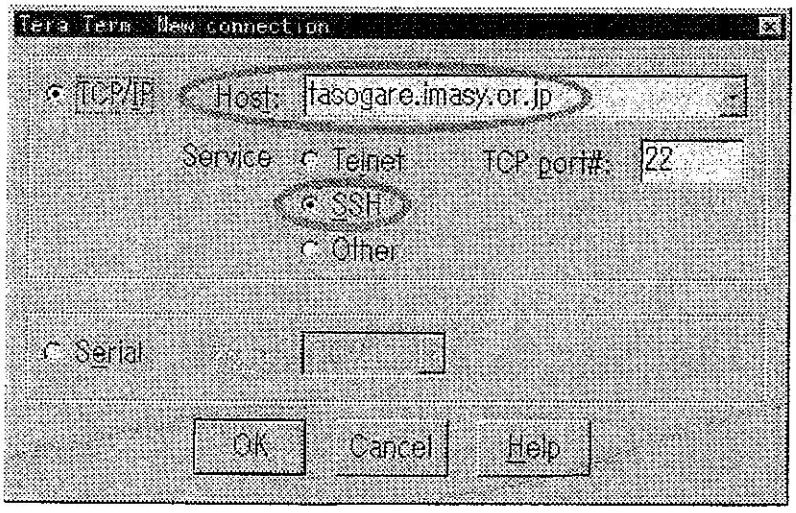


接続選択画面初期状態

見慣れたTeraTermの接続先選択画面に「SSH」という項目が加わりました。

ここで不幸にも、「Could not load TTXSSH.DLL！」というメッセージが出ていつもの TeraTerm の接続選択画面が開いてしまったらたら、まず ttxssh.dll が TeraTerm と同じフォルダにあるかどうかを確認して下さい。
ちゃんとあるのにも関わらず同じメッセージが出てしまう場合、irc クライアントの CHOCOA (http://www.forest.impress.co.jp/chat.html#chocoa) をインストールすると使えるようになる場合がありますので、試してみて下さい。

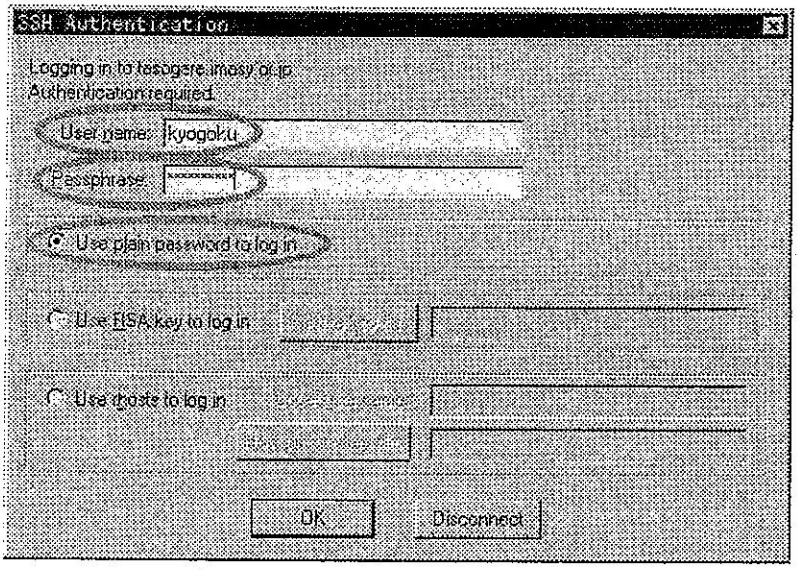
さて、ではネットワーク越しにログインしてみます。ttssh.exe を実行して下さい。



接続選択画面で tasogare を入力中

そして接続選択画面で Host:のところにログイン先のホスト名 (ここでは tasogare.imasy.or.jp) を入力し、Service で SSH のラジオボタンをチェックしてから「OK」ボタンを押して下さい。

暫くすると今度はユーザ認証画面が出ると思います。



ユーザ認証画面

ここで、User name にログイン先のアカウント名 (大抵はメールの宛先と同じ) を、Passphrase にログイン先のシェルログイン用のパスワード (telnet や ftp でログインする時に使うパスワード) を入力ます。
下のラジオボタンが「Use plain password to log in」にあることを確認し、「OK」ボタンを押して下さい。
ここではアカウント名「kyogoku」としてありますが、当然のことながら実際にはあなたのアカウント名をいれて下さい。またパスワードは何をいれても「*」と表示されますので、打ち間違いに注意して下さい。

目的のホストへログイン出来ましたか？

以上の手順でとりあえずネットの先へ ssh を使ってログイン出来る様になりました。
本物のパスワードを打ち込むので盗聴が心配ですが、ユーザ認証の段階で既に通信は暗号化されているので、ここて入力したパスワードは平文で送られるわけではなく安心です。

TTSSH ではパスフレーズを使った認証もサポートされていますが、それを使うための鍵生成ジェネレータが Windows で用意されてないのでここでは説明しませんが、適当な UNIX マシン等で鍵を作ってしまって、これを Windows へ持って来ることによって使うことが出来るようになります。
詳しくは TTSSH のヘルプを参照して下さい。

・Unix のシェルからリモートホストに ssh でログインする ～UNIX での SSH～

○ ssh のインストール

まずはローカルのシステムに ssh をインストールします。

適当なサイトから ssh を落として来て下さい。ちなみにオリジナルは、ftp://ftp.cs.hut.fi/pub/ssh/にあります。

場合によっては ssh2 の最新版 (ssh-2.0.10) を入れてしまってもいいかもしれませんが、ssh-2.x は

1) ssh-1.x とプロトコル的に互換性がない
2) 鍵交換等の設定が ssh-1.x と異なる
3) 利用制限が ssh-1.x に比べて厳しい

といったことに注意して下さい。
なお今回は ssh-1.x を使っての説明しか行いません。
ただし、ssh-1.x と ssh-2.x を同時にインストールすることは可能なので、あなたが個人的に ssh-2.x を導入することに全く問題はありません。
ちなみに、ssh-1.x でも IDEA(暗号化方式の１つ) の使用に関しては制限がありますが、個人で使用する分には気にしないでもよいでしょう。

さて、よほど変な環境でも無い限り、アーカイブを展開して

```
localhost% ./configure
localhost% make all
localhost% make install
```

とすればインストール出来ると思います。

FreeBSD であれば、ports を使うのも手ですね。

○具体的な手順

簡単な使い方としては、r系コマンドと同じ様に使います。

```
localhost% ssh remote.bar.org -l hogehoge
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? yes
Host 'remote.bar.org' added to the list of known hosts.
Creating random seed file ~/.ssh/random_seed.  This may take a while.
hogehoge@remote.bar.org's password:(ここであなたのログイン先のパスワードを入力)
Last login: Wed Nov 11 00:00:00 1998 from localhost
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.

FreeBSD 2.2.7-RELEASE (GONT) #1: Wed Oct 14 14:50:23 JST 1998
...
..
.

remote%
```

このように自動的に home の下に .ssh という directory が掘られて known_hosts と random_seed の2つの file が作られた後に、リモートホストの password を聞かれますので入力して下さい。
ちなみに、この時点で既に通信は暗号化されているので生 password を打っても大丈夫です。

さて、既に通信は暗号化されているのですが、それでも毎回生 password を入れるのも気持ち悪いので、ssh の特徴の1つである RSA 認証を使ってみましょう。

まずはローカルホストで公開鍵と秘密鍵を作ります。

```
localhost% ssh-keygen
Initializing random number generator...
Generating p:  ..........................++ (distance 414)
Generating q:  .........++ (distance 84)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (/home/hogehoge/.ssh/identity):(特に理由が無い限りここでは return)
Enter passphrase:(パスフレーズを入力。パスワードとは違います)
Enter the same passphrase again:(もう一度)
Your identification has been saved in /home/hogehoge/.ssh/identity.
Your public key is:
1024 35 153135988477098155..(中略)..63568537167572970497442989 1 hogehoge@localhost
Your public key has been saved in /home/hogehoge/.ssh/identity.pub
localhost%
```

これで、.ssh の下に秘密鍵 (identity) と公開鍵 (identity.pub) が出来ました。
次に今作った公開鍵を リモートホストにコピーします。
注意として、リモートホストへは必ず ssh でログイン、または直接コンソールへ入って作業して下さい。また折角なのでここで リモートホスト上で公開鍵と秘密鍵を生成 (つまり ssh-keygen を実行) してしまっても良いですが、以下では説明のためにそれは行いません。

```
localhost% ssh remote.bar.org -l hogehoge
hogehoge@remote.bar.org's password:(ログイン先のパスワード)
Last login: Wed Nov 11 00:03:00 1998 from localhost
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.   All rights reserved.

FreeBSD 2.2.7-RELEASE (GONT) #1: Wed Oct 14 14:50:23 JST 1998
...
..
.

remote% mkdir .ssh
remote% chmod 700 .ssh
remote% exit
logout
localhost% scp ~/.ssh/identity.pub hogehoge@remote.bar.org:~/.ssh/authorized_keys
hogehoge@remote.bar.org's password:(ログイン先のパスワード)
authorized_keys           |          0 KB |  0.3 kB/s | ETA: 00:00:00 | 100%

localhost%
```

ここでの注意として、既に authorized_keys ファイルがある場合には直接 scp せず、別の名前でコピーしておいてから、既存の authorized_keys ファイルに追加する様にして下さい。
また、scp なんて面倒だからカット&ペーストで書き込もうという方もいるかも知れませんがが、「identity.pub ファイルの中身は１行である」ということに充分注意して下さい。
さて準備が出来ましたので、早速 リモートホストへ ssh してみましょう。

```
localhost% ssh remote.bar.org -l hogehoge
Enter passphrase for RSA key 'hogehoge@remote.bar.org':先程のパスフレーズを入力)
Last login: Wed Nov 11 00:05:00 1998 from localhost
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.   All rights reserved.

FreeBSD 2.2.7-RELEASE (GONT) #1: Wed Oct 14 14:50:23 JST 1998
...
..
.

remote%
```

これで生パスワードを使わずにリモートホストへログインできるようになりました。

ローカルホストで、/usr/local/sbin/sshd を実行して ssh サーバを上げた後に同じ事を自分のホストに対して行えば、今度はリモートホストから自分のホストへ ssh 出来るようになるはずです。

今日からこれで何処へ行くにも ssh ですね。

○リンク

ssh について、より詳しく知りたい時は以下の FAQ を参考にするとよいと思います。

・日本語 ssh FAQ http://www.vacia.is.tohoku.ac.jp/~s-yamane/FAQ/ssh/

GIMP の Script-fu で描くフラクタル画像

fujita@mma.club.uec.ac.jp

0. GIMPって? ～この原稿が出来るまで～

今をさかのぼること数ヵ月前。MMA に念願 (?) の Alpha マシンが配置された。程なく NetBSD を軸にそれなりの環境[1] が整えられ、そろそろ MMA 内の端末の一つとして機能しようかという時期に uirou さんが GIMP v1.0.0 をインストールした[2]。使ってみるとこれがなかなか良くできたもので、外観・機能なんかは Adobe の Photo Shop を想像すると大体合っている[3]。それまで X 用のペイントソフトというと Xpaint くらいしか見たことがなかったのでその機能のおもしろさに興味を持ち、喜々として使ったもんである。とは言え、GIMP を使うなどという奇特な人間は日本には少ないらしく、ユーザーマニュアルも一部日本語化されてはいるものの、重要な知識はそうそう容易には手に入らなかった。だから GIMP の用途は専らマウスを使ってぐりぐり絵を描くことだけだった。そうこうしてるうちに夏休みが終わり二学期が始まって、そろそろ調布祭という時期がやってきた。百萬石のネタを考えねばならない。それとほぼ時を同じくして、GIMP ユーザーマニュアル日本語版の Web ページに Script-fu の書き方が掲載されているのを発見。「これは Script-fu のことを原稿にしろ」という神の啓示に違いない。つーわけで Script-fu と戦うことになったのでありました。

1. Script-fu というもの

Script-fu は Scheme という Lisp 似の馴染みの薄いインタプリタ言語で書かれた GIMP 用のスクリプトで、それなりに細かい作業も出来たりする。Lispっぽいから 1+2 は (+ 1 2) と書くし、リストという概念も存在する。何しろ一命令につき括弧が一組みくっつくので、書いている内に括弧だらけになって混乱することも多いのも事実だ。Script-fu で使える GIMP 特有の関数の情報 (説明、引数、戻値、etc.etc.) は、GIMP のメインメニューの XTNS の下、DB Browser を用いて見ることが出来る。それらの関数を使用して様々な挙動、文字を白抜きに加工するとか幾何学模様を描くなど、手作業で行うには煩雑な一連の動作過程を記述できるのが Script-fu の利点である。さて、Scheme の文法は他の文献を当たってもらうことにして、次に実際のソースを見てもらいましょう。百聞は一見に如ず、というヤツです。

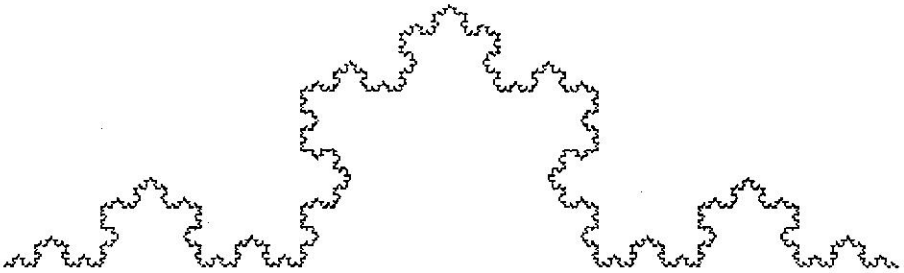
2. 実例 1: 楽しい Koch 曲線

Script-fu を書くにあたって、さて何をやったものかと一瞬悩んだ挙げ句に出た結論がフラクタル画像の描画だった。これならばっと見がおもしろいし、何よりアルゴリズムが知れてるから大した苦労もなく出来そうだ、と思ったわけで。実際やってることとは数値計算であって、かろうじて GIMPっぽいのはペンシルで線を引く所だけだったりします。

```
; コメントはセミコロン以降の行末まで。
(define (draw-line img drw a b c d n) ; 二点 A(a,b),B(c,d) を結ぶ線分を描く関数
    (if(= n 0)                        ; if 文も当然あります。
      (begin                          ; (let* ) で囲まれた部分は局所変数になる
        (let* ((vec (cons-array 4 'double))) ; 4 つの要素をもつ配列 vec を確保。
              (aset vec 0 a) (aset vec 1 b) (aset vec 2 c) (aset vec 3 d)
          (gimp-pencil img drw 4 vec))) ; ペンシルで線を引っ張る。
      (begin                          ; n!=0 のとき
```

[1] home のマウントとか tcsh とか ssh とか less とか
[2] ということは、uirou さんが GIMP を入れなければ、この記事は無かったのである。多謝> uirou さん
[3] てゆーか、そのまんま :p

```
        (let* ((length (sqrt (+ (* (- a c) (- a c)) (* (- b d) (- b d)))))
               (cos-theta (/ (- c a) length)) ; 線分 AB の長さ length を計算し、
               (sin-theta (/ (- d b) length)) ; X 軸との正弦・余弦を出し、
               (s1 (/ (+ (* 2 a) c) 3))       ; AB を 3 等分する点 C(s1,t1) と
               (t1 (/ (+ (* 2 b) d) 3))       ; D(s2,t2) を計算し、
               (s2 (/ (+ a (* 2 c)) 3))       ; 2 点 CD を一辺とする正三角形の
               (t2 (/ (+ b (* 2 d)) 3))       ; 頂点 E(x,y) を出し、
               (h (/ (+ a c) 2))              ; 線分 AC,CE,ED,DB それぞれに対して
               (i (/ (+ b d) 2))              ; 再帰的に同じ計算を繰り返す。
               (x (+ h (* (/ length (* 2 (sqrt 3))) sin-theta)))
               (y (- i (* (/ length (* 2 (sqrt 3))) cos-theta))))
          (draw-line img drw a b s1 t1 (- n 1))
          (draw-line img drw s1 t1 x y (- n 1))
          (draw-line img drw x y s2 t2 (- n 1))
          (draw-line img drw s2 t2 c d (- n 1))))))
                                              ; メインルーチン
(define (script-fu-draw-Koch-curve image draw x1 y1 x2 y2 depth)
    (gimp-image-disable-undo image)           ; undo を出来なくして、
    (set! old-brush (car (gimp-brushes-get-brush))) ; 現在の Brush の状態を保存し
    (gimp-brushes-set-brush "Circle (01)")    ; 一番小さい Brush に変えて、
    (draw-line image draw x1 y1 x2 y2 depth)  ; 線を引く。
    (gimp-brushes-set-brush old-brush)        ; 諸々の状態を元に戻す。
    (gimp-image-enable-undo image)
    (gimp-displays-flush))                    ; 画像をフラッシュして終わり。
  ; register には GIMP に登録する情報を書く。これは DB Browser で見ることが出来る。
(script-fu-register "script-fu-draw-Koch-curve" ; 最初はメイン関数の名前
  ; 2 番目はどこに script を置くか。この場合、画像を右クリックしたときに出て来る
  ; メニューの Script-fu 以下のところに"Koch Curve"という項目が出来る。
                "<Image>/Script-Fu/Render/Koch Curve"
                "Draw a Koch Curve"          ; script の説明
                "Fujita Takeo"               ; script の作者
                "fujita@mma.club.uec.ac.jp" ; script の著作権
                "1998/11/5"                  ; script の日付
                ""        ; 取扱える画像の種類。Indexed とか Gray Scale とか。
  ; 以上の 7 つが必須の項目で、以下のものは script に与える値。
                SF-IMAGE "image" 0 ; IMAGE は描画するイメージの値らしい
                SF-DRAWABLE "drawable" 0 ; DRAWABLE はレイヤの値らしい
                SF-VALUE "x1" "0.0"      ; 以下のものは、線分の始点と
                SF-VALUE "y1" "200.0"    ; 終点の座標、それと繰り返しの
                SF-VALUE "x2" "200.0"    ; 回数で、右の値がデフォルト値。
                SF-VALUE "y2" "200.0"
                SF-VALUE "depth" "5")
```



出来上がり:段々曲線に近付くから Koch 曲線という

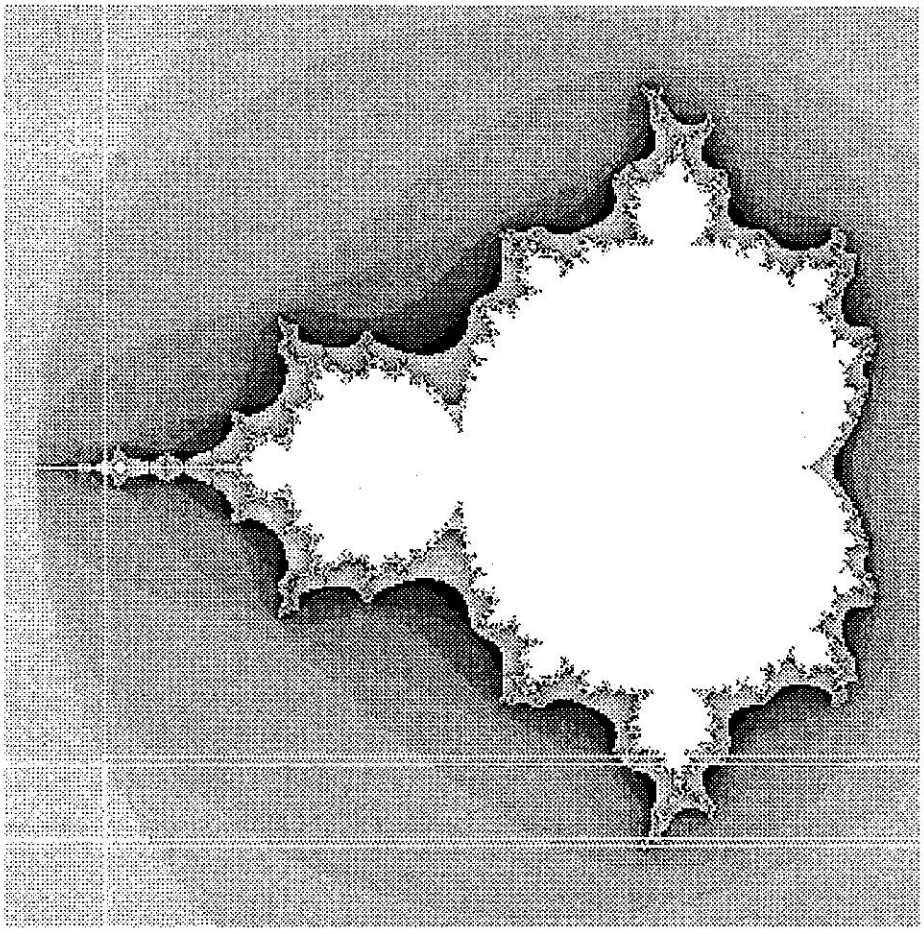
3．実例 2: 楽しい Mandelbrot 集合

フラクタルの代名詞、マンデルブロ集合なんですが、実はソースを書いた本人はこのアルゴリズムを理解できていません。どうしてこう計算するとあの変な絵が出来上がるのか、全くもって分っていないのです。複素数平面上で $z_n = f(z_{n-1})$ という漸化式を解くとこーなるらしいんですが。

```
(define (color-set color a0 a1 a2 a3 a4 a5 a6 a7) ; 色の配列を作る関数
    (aset color 0 a0) (aset color 1 a1) (aset color 2 a2)
    (aset color 3 a3) (aset color 4 a4) (aset color 5 a5)
    (aset color 6 a6) (aset color 7 a7))
                                                  ; メインルーチン
(define (script-fu-draw-mandelbrot-set image draw sr si er ei iter)
    (set! height (car (gimp-image-height image))) ; 画像の高さと幅を取る。
    (set! width (car (gimp-image-width image)))
    (set! dr (/ (- er sr) width))                 ; 実部の増分
    (set! di (/ (- ei si) height))                ; 虚部の増分
    (set! red (cons-array 8 'double))             ; 使用色の指定
    (set! green (cons-array 8 'double))
    (set! blue (cons-array 8 'double))
    (color-set red 223 191 159 127 95 63 31 0)
    (color-set green 223 191 159 127 95 63 31 0)
    (color-set blue 223 191 159 127 95 63 31 0)
    (set! old-brush (car (gimp-brushes-get-brush))) ; 現在の状態の保存
    (set! old-color (car (gimp-palette-get-foreground)))
    (gimp-image-disable-undo image)
    (gimp-brushes-set-brush "Circle (01)")
    (let* ((cr sr))              ; あとはひたすら数値計算
      (while (<= cr er)          ; ちなみに今まで度々出て来た (car ) というのは
        (let* ((ci si))          ; 例えば '(1 2 3) というリストがあったときに
          (while (<= ci ei)      ; (car '(1 2 3)) の出力としてリストの先頭の
            (let* ((k 0)         ; 要素を返す関数である。
```

```
                (zr 0) (zi 0)
                (flag 1))
            (while (and (<= k iter) (= flag 1)) ; 実はscheme にはfor文がない
              (let* ((r (+ (- (* zr zr) (* zi zi)) cr))
                     (i (+ (* 2 (* zr zi)) ci)))
                (if (> (+ (* r r) (* i i)) 4)
                  (begin
                    (let* ((c (fmod k 8))
                           (plot (cons-array 2 'double))
                           (pixel-color (cons (aref red c) \
                             (cons (aref green c) (cons (aref blue c) ()))))))
                      (aset plot 0 (/ (- cr sr) dr))
                      (aset plot 1 (/ (- ci si) di))
                      (gimp-palette-set-foreground pixel-color)
                      (gimp-pencil image draw 2 plot)) ; ここで点を描画してる
                    (set! flag 0))
                  (begin (set! zr r) (set! zi i))))
              (set! k (+ k 1))))
          (set! ci (+ ci di))))
      (set! cr (+ cr dr))))
  (gimp-palette-set-foreground old-color) ; 状態を元に戻して終わり
  (gimp-brushes-set-brush old-brush)
  (gimp-image-enable-undo image)
  (gimp-displays-flush))

(script-fu-register "script-fu-draw-mandelbrot-set" ; 登録用のregister
                    "<Image>/Script-Fu/Render/Mandelbrot Set"
                    "Draw Mandelbrot Set"
                    "Fujita Takeo"
                    "fujita@mma.club.uec.ac.jp"
                    "1998/11/9"
                    ""
                    SF-IMAGE "image" 0
                    SF-DRAWABLE "drawable" 0
                    SF-VALUE "SR" "-2.1"  ; 初期値は二点の虚数の実部と
                    SF-VALUE "SI" "-1.35" ; 虚部だそうです...
                    SF-VALUE "ER" "0.6"
                    SF-VALUE "EI" "1.35"
                    SF-VALUE "ITER" "10")
```



出来上がり:ヒョウタン

4．参考文献

(1) The GIMP User Manual [ja]　　http://pcv.awa.tohoku.ac.jp/

(2) SIOD: Scheme in One Defun　　http://people.delphi.com/gjc/siod.html

(3) 生協で立ち読みしたフラクタル画像の本

ssh over HTTP

uirou@mma.club.uec.ac.jp

1998/11/19

概 要

前、ssh over HTTP な soft を試作、実験してみたことは前回の百萬石にて報告した。まぁ、要するに HTTP を使って remote host の ssh port を local host にもってきてしまおうという物である。で、事件は突然やって来る。

ある日、偉い[1] 人は言った。「ssh over HTTP のソフトってくれますか?」これだけならば、ほいほいと言って source code を渡したであろう。でも、その後にこう、続いていたのである。「あの... SUN の人が使いたいって言ってるんだけど、良い?」ととと、とんでもない。あれって、「わーい。動いた。あー、満足。満足」で、ほったらかしてあったへなへなな code なのである。

こりゃ困った。とてもとても身内以外にみせられるような code じゃないぞ？http encode しないでも通る所でしか使ってなかったから、http encode はしてないし、なんだか謎な所で落ちる事があるし。で、結局、断ってしまった[2]。

で、今回、百萬石を書くに当たって、ふ、と思い出したので書き直してみる事にした[3]。

1 これはなに？

ssh[4] という便利な soft がある。これは、何処か遠くの host に login する時に、間の通信を暗号化してくれる便利な soft なのだが、相手の host に直接 tcp でお話が出来ないと login することが出来ない。要するに、MMA のように相手が local network にあったり、自分が local network の中にいる場合などがそうである。

で、気弱な僕としては偉い人[5] に「あれこれの host に ssh したいから global IP くれ」とか、「そこここの host に ssh したいから global IP なマシンにアカウントをくれ」などとは言えないのである。

でも、最近では http proxy でもって外の情報に access するような事は出来るようになっている物である[6]。実はこれを使えば data の送受信が出来てしまうのである。

ということで、その HTTP を使って目的とする相手の host の ssh port を持って来て、local な port に繋げてしまうものを作ったわけである。

とりあえず名前が無いのは可哀想なので、“webpipe” と名付けてあげる事にした[7]。

2 出来る事

webpipe-2.0.0 には以下の機能が実装されている。

- http proxy を介した remote httpd への通信

[1] そう，とっても偉い
[2] ごめんなさい。
[3] そう，結局の所前回の焼き直しだったりする
[4] http://www.cs.hut.fi/ssh/
[5] network の管理者様とか
[6] 何処ぞの OS では OS に組み込まれている位だし (?)
[7] もっと良い名前を募集中である

- http proxy を介さない remote httpd への通信
- remote httpd から connect 出来る (another)host への通信
- ssh port 以外の port への通信
- password auth を用いた httpd への connect

3 動作原理

webpipe は以下のような方法で remote host からの通信を実現している。

client からの送信は、POST を用いて remote httpd へ送られる。remote の socket からの送信は nph-script を用いて local へ送られる。

これだけである。

4 出来ない事

webpipe には今後以下の機能を実装する予定[8] である。

- keep-alive を用いた通信の高速化
- zlib を用いた通信の圧縮
- SSLay を用いた通信の暗号化

後半に行くにしたがって「出来たら良いな」の度合が高まる。

5 お前、ムカつくんだよ

これだけでは単なる soft のレビューになってしまうので、作った人らしい事でも書いてみる。

今回の書き直しは「きれいな source」にするのが目的であった。しかし、僕が「きれいな source」なんて書けるわけが無い。ということでこの点はあきらめた。そこで、「ムカつくところを直す」ことにした。

5.1 HTTP って POST でも encode しないといけないんじゃない？

言い訳を書いていると長くなるので書かない[9] が、僕は RFC を真面目に読んでいない。よって POST method において送信される data は「gif file が binary でてろてろ流れて来るんだから、POST なら encode しないでもいーだろー」とか、考えていた。だが、そうでもないらしい。よくある CGI のサンプルでは、GET だろうが POST だろうが、decode してから data として使用するのである。

あちゃー。まずいよ。これは。http encode してから POST しなきゃ駄目じゃん。

ということで、直した。

5.2 remote の ssh port 以外にも接続できると良いよね

最初は ssh をしたかっただけだったので localhost:22 しか接続できなかったのだが、相手の host がその httpd が動いている host だとは限らないし、ましてや port が 22 だという保証は何処にもない。じゃぁ、っ

[8] 言い替えれば実装できてない
[9] 単に英語が読めないだけである

てんで、その httpd が動いている host から接続できるすきな host のすきな port に接続できるようにしよう、ということで、した[10]。

5.3 fgets(3) は使いたいけど、select(2) が使えなくなるんだよね

長く perl(1)[11] の世界に浸かりすぎていたからだろうか、入力される data を一行おきに処理すると言った code を書きたくなる。でも、それを実現してくれる fgets(3) ライブラリルーチンは、FILE 構造体というものを利用しないといけない。しかし、FILE 構造体を用いると、内部でバッファされてしまうので select(2) システムコールが意味をなさなくなってしまう。ムカつく。

ということで、read だけ buffer して、fgets や select の出来る FILE 構造体の紛い物を作った。

5.4 HTTP の password auth って、ツルツルなのね

httpd による authentication に、password を用いた物がある。これは、単純に "username:password" を base64 encode して HTTP headder に混ぜて送るだけのようである。こんな感じである。

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

ということで、実装した。

以上である。

6 似たような program って、無いの？

世間は広く、似たような事を考える人はいるもので、ruby[12] で動作する HttpTunnel という soft が存在する。しかしこちらは、相手の httpd として、自前の program を動かすので、webpipe よりもうちょっといろいろな権限が必要だったり、webpipe よりも高速だったりしそうである。

7 謝辞 (なのか？)

本文中、とっても偉い人からの mail の一部を無断で引用しました。

参考文献

RFC1945 Hypertext Transfer Protocol – HTTP/1.0

RFC2068 Hypertext Transfer Protocol – HTTP/1.1

[10] これで telnet も POP も一応通るぜ。暗号化してないけど。:P

[11] http://www.perl.com/

[12] http://www.netlab.co.jp/ruby/jp/

OSのリモートインストール

君島 秀征 <sanjigen@mma.club.uec.ac.jp>

平成10 年 11 月 21 日

概 要

ネットワーク上に存在するマシンのOSを、物理的に手を触れずに入れ換えることは可能でしょうか。ここでは、それが実現できるかどうかを考えてみます。

1 発端

調布祭を目前に控えた11月13日、MMAの部会がありました。調布祭に関する議事が主でしたが、それ以外にも、先日取得したmma.gr.jpドメインをどのように運用するかという議題もありました。mma.gr.jpのマシンをネットワーク上のどこかに置かなければなりませんが、MMAですからマシンをリモートで管理したいというのは自然の欲求です。

プロバイダ側でLinuxマシンをNOCに用意してくれて、したいことをさせてくれるサービスを提供しているところがあるそうで、それを利用するのが手っ取り早そうです。

しかし、mma.gr.jpドメイン取得プロジェクトの中心人物が、Linuxは嫌いだと言っています。ネットワーク上からFreeBSDを入れ換えられないでしょうか。

新しいOSをddして再起動すればいいじゃん。

2 本当にできるのか?

2.1 新しいOSをdd

幸い、Linuxはスワップ領域にスライス[1]を一つ割り当てているので、スワップ領域にFreeBSDのmini rootをddしてやって、そこから再起動させれば良いでしょう。あとは、mini rootから起動したFreeBSDの上でfdiskやdisklabelやnewfsをしてMAKEDEVして、配布ファイルを展開していけばいいわけです。

スワップ領域にddする前に、スワップを使わない設定にしてLinuxを再起動させる必要はあります[2]。

これが、FreeBSD⟶LinuxのOS入れ換えだと、少し面倒くさそうです。もしHDD全体をFreeBSDが使っていたりすると、disklabelやfdiskと戦ったりしなければならないでしょう。/usrのバックアップを取ってumountしてdisklabelをいじってパーティションを小さくしてnewfsしてmountしてバックアップを戻し、その後fdiskでFreeBSDスライスを小さくする、というようなことが必要でしょうか。

2.2 再起動

再起動させるとまたLinuxが立ち上がってしまうのでは、というツッコミが入りましたが、次回起動時の起動スライスはMaster Boot Record(以下、MBR)に書かれているだけなので、それを書き換えてしま

[1] パーティションとも言います。ひとつのHDDに4つまでしか作れない領域のことです。
[2] 一度swaponしたら、終了するまで解放できないはずです。

えばいいだけですね。

Linux ですから、LILO の config を書き換えるという手もあります。

FreeBSD の fdisk ですと、`echo 'a 2' | fdisk -f - wd0` などとすることで起動スライスを設定できます。この場合ですと、wd0s2 から起動することになります。Linux の fdisk でも起動スライスを設定するくらい、できますよね。

2.3 mini root

mini root を入れたいスライスの容量以下のイメージを作らなければなりません。ローカルマシンに環境を構築したものを送って dd します。ローカルマシンで厳密にテストした方が良いでしょう。

FreeBSD の普通の最小構成に root のパスワードを設定して sshd が動くようにして... たぶん、容量は 100MB も要らないでしょう。

3 結論

本当にできそうですね。困ったなぁ。

4 最後に

言い出した私が実際にやってみろ?

毎年恒例のインストール大会として、1 年生にやらせてみるというのはいかがでしょうか。

SWORD Emulator を作る

tree (tate@spa.is.uec.ac.jp)

平成 10 年 11 月 21 日

1 はじめに

読者の皆さんは S-OS SWORD を御存知だろうか? では、Oh!MZ は? ええい、Oh!X でもいい。知らない人は WEB の検索エンジンに+"sword" +"s-os" +"MZ" あたりを放り込めば情報が得られるだろう。便利な時代だ、使わない手はない。

S-OS は月刊誌 Oh!MZ (後に Oh!X に誌名変更し、現在は不定期 MOOK) 上に掲載された、Zilog Z80 を用いたパソコン用の共通 OS である。OS といっても殆ど BIOS のエントリルーチンと仕様を共通化しただけに近いものだが、MZ-80K を含めた Z80 搭載の MZ シリーズ及び X1 シリーズを始めとして、PC-8001/8801, PASOPIA, SMC-777C, FP-1100, MSX など非常に多くの機種に移植された。また、「S-OS で一度書いたプログラムはどのマシンでも動く」という最近よく聞く宣伝文句もあり、非常に意欲的な開発が行なわれ、開発環境のみならず、ゲーム等が数多く発表され、実数演算や 3D グラフィックスも拡張パッケージとして利用可能となっていた。SWORD は S-OS の 2 番目の実装であり、実質的なリファレンス実装である。

時は過ぎ。Z80 の扱えるメモリ空間 64K バイトは CPU の L1 キャッシュよりも小さくなり、大容量 2D floppy の 320K バイトも CPU コアに乗る時代になった。

一方で Z80 の時代にすら魔神語と呼ばれた機械語は、パイプラインやアウトオブオーダーを意識したものとなり、計算機の助けを得て記述するものになった。

計算機を使う遊びは高度になり、肩肘を張ったものが増えてきた。温故知新と云うように、たまには振り返ってみよう。

と、いうことで、UNIX 上での S-OS SWORD Emulator の製作である。

2 設計

2.1 基本方針

S-OS は特定のハードウェアを想定しているわけでなく、基本的にシステムコールの集合である。従って Z80 CPU のエミュレータに、それらのシステムコールを実装してやれば良い。

SWORD のシステムコールは 53 個であり、その殆どは入出力命令である。

SWORD の基本入出力メディアは 2400bps のカセットテープであるが、今回はカセットインターフェイスは割愛し[1]、ハードディスク上にフロッピーディスクのイメージを作成する方法と、UNIX 上のファイルシステムに直接アクセスする方法の二つを実装した。

[1] サウンド I/F を用いたカセットインターフェイスの実装は実現可能であり、既に他の実装が存在する

ディスクイメージファイルを作成する方法は、「生」のフロッピーにアクセスするようなアプリケーションでも動作させることができるうえ、メディアコンバートもディスクイメージをコピーするだけなので手軽である。しかし、UNIX 側から個々のファイルにアクセスできないのは不便である。

UNIX 上のファイルへのアクセスには、SWORD にけるカセットへのシーケンシャルファイルの保存/読み出しを対応させる。この際、SWORD においてはファイルアトリビュートとしてフィイルモード、格納アドレス、実行開始アドレス、ファイルサイズが必要となるので、固定長のヘッダをファイルの先頭に付加することにより、これらの情報を保存する。さらにアスキーファイルに関しては改行コードの変換を行なうことにより、SWORD 用アスキーファイルを UNIX 上のエディタでそのまま編集することを可能とする。

2.2 Z80 エミュレーション

幸い Z80 のエミューレータは数多く存在する。今回はその中で CP/M エミュレータの yaze の Z80 エミュレータを用いた。yaze のパッケージ内の simz80.c は適当なメモリ空間とレジスタの情報を与えて関数 simz80() を呼び出すことにより、HALT 命令に到達するまで Z80 コードの実行を行なってくれる。HALT 命令に到達すると、この関数は HALT 命令の次の PC (Program Counter) の値を返す。この時の PC が指し示すメモリの値をパラメタとして利用することにより、Z80 の HALT 命令を (UNIX 側の native code で動作している) SWORD エミュレータへのシステムコールとして利用することが可能である。

2.3 コードの切り分け

重要な設計項目として、Z80 エミュレータから SWORD エミュレータに制御をどこで移すかがある。

当初は SWORD の各システムコールを直接 SWORD エミュレータのシステムコールを呼びだす形で設計し、試験的に実装したのだが、SWORD は他の実装がないのと、ソースコードが公開されていたため、SWORD の内部情報や仕様外の動作を利用したアプリケーションが数多く存在し、それらが正常に動作しなかった。

これに対処するため、SWORD の基本部分は Oh!MZ 誌に掲載された Z80 用のコードをそのまま利用し、各機種用のモジュールへの分岐部分で SWORD エミュレータに制御を移すように設計した。

また、いくつかのシステムコールは機種別 BIOS の変数領域を参照するため、単純な MZ-80K を模したメモリ構成とした。

3 実装

3.1 移植性の向上

実装は BSD/OS 上の C 言語で行なった。移植性を高めるため、以下の規則に従って実装した。

- ANSI, POSIX に準拠した関数を使う
- 無ければ BSD の関数を使う
- 対象 OS に BSD 関数が無い場合には、これを作る

今回はコンソールベースの実装としたが、ターミナル I/O に termcap を用いているなど、UNIX に依存した実装になった。

3.2 モジュール分割

今回の実装では、以下のモジュールに分割して実装した。

simz80　Z80 エミュレータ (yaze より)

sos　メインルーチン

trap　システムコール管理

dio　DISK I/O ルーチン

screen　コンソールコントロール

util　ユーティリティ関数群

compat　システムに不足する関数群

メインルーチンは Z80 のメモリ空間を適当に初期化し、SWORD エミュレータ全体を制御する。Z80 エミュレータが HALT 命令を実行すると、制御はシステムコール管理部に移り、DISK I/O ルーチンやコンソールコントロールルーチンを利用して、実際の I/O 処理が実行される。

特にコンソールコントロール部は内部に仮想画面を持ち、これをシステムコール終了のタイミングで実画面に反映させている。

4 評価

コンソール画面を用いたゲームや、スクリーンエディタ、言語等は動作を確認してる。

速度は 486 クラスのマシンでも一般的な Z80 マシンを凌駕する。此我の性能差を考えれば当然であるが、画面描画だけは複雑な画面制御を行なうために、特に X Window System 上で動作させた時にはさほどでもない。

現在までに SWORD エミュレータの動作した OS は、FreeBSD, BSD/OS 2.1, NetBSD, Linux, SunOS 4.1.3, Solaris 2.5, NEWS-OS 4.2.1, NEWS-OS 6.1, IRIX 5.3, BeOS for Mac などである。ただし、BeOS に関しては一部の機能に制限がある。

5 改良すべき点

5.1 画面描画の高速化

特に xterm 等のターミナルエミュレータで動作させた場合、画面描画が大きなボトルネックになる。

これに対処する方法として、仮想画面の実画面への反映を SWORD のシステムコール終了時ではなく、タイマ割り込みによる定期的な反映にすることが考えられる。

これは実際に実験的に実装を行なった。1/30 秒毎に反映をするよう実装したが、Z80 エミュレーションによる仮想画面の書き換えがあまりに高速なため、実画面に表示される前に次の画面が表示されるという現象が生じ、一部アプリケーションが使いものにならなくなった。

一例をあげれば、インベーダーゲームにおいて、ゲームを開始した次の画面では自機に敵ミサイルが到達している状況になった。

5.2 X 化

SWORD の期待するキャラクタセットは MZ のものを基本とするため、多少特殊である。現在はこれらのキャラクタセットを持たないため、一部のキャラクタは正常に表示されない。

また、リアルタイムキー入力やグラフィック処理の追加など、X Window System を利用すべき要因は多い。

これらに関しては、折をみて対応する予定である。

5.3 互換性の向上

SWORD においては、仕様書には記述されていないが、数多くのアプリケーションが利用している「常識」が存在した。

現在の SWORD エミュレータはこれらの「常識」を一部実装していないため、これらを実装して互換性を高める必要がある。

5.4 移植性の向上

現在の多くの OS は POSIX に準拠しているため、基本部分の移植は比較的容易である。BeOS への移植はその好例であろう。

しかし、termcap や拡張された termios に代表される UNIX に依存したコードは移植性を低めてしまっており、Windows などへの移植などの妨げとなっている。

これらの OS 依存性の高いコードは、主にコンソールコントロールモジュールに集中している。そこでコンソールモジュールをさらに、仮想画面を制御する OS 依存性の低い部分と、実際の画面描画等を行なう OS 依存性の高い部分とに分割し、後者を OS 毎に用意する方法が考えられる。

これは前述した X Window System 対応においても有効な分割であり、実施を予定している。

6 配布について

Z80 エミュレータ部分を利用した yaze は GPL に従って配布されており、エミュレータ本体の配布に関しては殆ど問題はない。

しかし、SWORD エミュレータの実行には Z80 コードで記述されたオリジナルの SWORD バイナリが必要となるため、現在は一般配布は行なっていない。

7 おわりに

この時代に SWORD エミュレータなど実用性は殆んど皆無である。

制御用マイコンなどの開発には CP/M エミュレータや MS-DOS 上のクロス開発環境の方が整備されているだろう。過去のゲームがやりたければ、fMSX などの各機種用エミュレータの方が楽しめるだろう。

しかし所詮は趣味である。普段は画面の装飾に忙しい CPU を、懐しいアプリケーションに分け与えたところで罰が当たることはあるまい。

termに壁紙を貼ってみやう

uirou@mma.club.uec.ac.jp

1998/11/20

この programs を書いた時にカンジワルイ応対をしてくれた全ての MMA 部員に捧げる

1 要するに、それって、壁紙じゃん

ある日、oolongが言った。「oclockの背景に絵を張り付ける改造をしているんだけれど、backgroundPixmapっていうresourceを使うと凄く簡単だよ」要するに、Athena Widgetにはそういうresourceがあり、それを使うとXClearArea(3X11)[1] するだけでそのpixmapを表示させる事が出来るというのである。sourceを読ませてもらう。確かに、凄く簡単。あ、要するに、それって、壁紙じゃん。

2 じゃぁ、Athena Widgetなら、なんでもなの？

へー、すげー。じゃぁ、Athena Widgetを使ってるアプリなら、そいつのresourceにpixmapを指定してやれば、壁紙アプリになっちまうってこと？と、いうことでxload(1)と戦ってみた。なぜならば、xloadはstripChartWidgetClassを使っているだけであり、そいつのresourceにも同じbackgroundPixmapが存在していたからである。で、一つの問題点[2]を除いてなんなく完成。

おつぎはxdm(1)である。こいつはちょっと面倒臭かった。xdmは文字列を表示したり、消したりしていて、その時にXFillRectangle[3] とかで消す処理をしていたので、ただ、backgroundPixmapに適当なpixmapを張り付けるだけでは終らなかった。まぁ、これはXFillRectangleをXClearAreaに書き換ただけで問題なし。これも完成[4]。

ここらあたりで、壁紙というものが非常に面白い物であると気がついてしまったのでAthenaWidgetを使った色々なsoftを壁紙化できるかなー、と思い、よくよく調べてみるとどうもXSetBackgroundPixmap(3X11)というfunctionが用意されていて、これはAthenaWidgetの機能ではなく、Xlib、要するにXの機能であるらしい事が解った。

3 KTermの壁紙patchってのがあったよねー

へーへー。すごーい。じゃぁ、何でも壁紙化出来るじゃん。そういえばKTermの壁紙patchってのがあったよねー[5]。あれって、どうやっているんだろう？ということで持って来る。patchを読む。

えぇー、これって、自前でpixmapを持って、いちいち計算して転写してそのうえに文字を書いてるのー？

---

[1] 要するにclear
[2] loadのグラフが横にずれるのは実際にずらしていただけなので、そこをrefreshに書き換えただけ
[3] 一色で塗りつぶすfunction
[4] 現在MMAで使用しているのはこいつである。そういえば、一つ致命的なバグが報告されてたなぁ、、
[5] http://karin.isl.titech.ac.jp/%7Etakagi/kterm/

なんか、それって、遅い気がする。ということで改造を開始。元祖のKTermの壁紙patchを元に、XSetBackgroundPixmapを利用した物に書き換えて行く。結構大変。X[mb—wc]?Draw[Image]?String[16]?とかがいろいろ悪さをする。特にDrawImageString系。これは背景を単色で塗りつぶすので壁紙には全く適さないのである。しかし、それを全てDrawString[6]に変えれば良いと言う物でも無かった。DrawImageStringは、文字の反転に使われていたのだ。反転した文字を背景を単色で塗りつぶさないと、本物との整合性が取れないし、第一、白地の壁紙に黒の文字が反転して白地の壁紙に白で書かれては何が書いてあるか読めない。これじゃぁ使えないのでその部分のDrawImageStringはそのまま残す必要があった。この問題がけっこう理解するまでに時間がかかった。けれど、本家が既にほとんどやってくれていたのでその真似をすればOKであった。

ということでKTermのXSetBackgroundPixmapによる壁紙化も完成。本家と比べて、僕の使っているX[7]では約4倍の速度向上がみられた。満足。

4 壁紙が好きな時に書き換えられないなんて、嫌だ

しばらくそのままで壁紙KTermを使用していたのだが、そのうち、壁紙がcommandを実行した時だけにしか選べないで、実行させた後で変えられないなんて、ヘナヘナじゃん。とか、思うようになった。だって、XSetBackgroundPixmapをもう一度発行すれば、背景の貼り替えが出来るはずだからである。

そこで、XGetInputFocus(3X11)を使用して現在focusされているWindow(要するにcommandを入力したterm window)のBackgroundPixmap属性を書き換えるprogram(WallPaperChanger: wpc)を作った。これは結構(自分的に)ヒットであった。なにせ、

```
#!/bin/csh -f
while(1)
  foreach foo ( dir/dir/*.xpm )
    wpc $foo
    sleep 300
  end
end
```

とか仕掛けておけば、気がついた頃にtermの壁紙がさくっと入れ替わるのである。

5 Xpmしか使えないなんて、画像の明るさが変えられないなんて、嫌だ

また、初期のwpcではXpmしか使えなかった。このXpmという画像formatはけっこうHDDを食う。なにせ、データの中身が全てASCII、それも人間が読める形式でかかれているのだ。これでdataがでかくならないわけが無い。それに、Web上で何かぐっとくる画像[8]を見つけて来たとしても、それをいちいちXpmに変換してからでないと使えなかった。

そんなの嫌だ。第一面倒臭い。それにHDDを食われるのはただでさえHDDの少ないnoteをmainな環境としている僕には辛すぎる。それにXpmだと256色しかsave出来ないじゃないか。そんなタコな話があるか。いや、ない。ということで、wpcにjpg,png,gifのloadの機能もつけた[9]。ついでに、それぞれ

[6] 文字だけを描画するfunction。背景は塗りつぶさない
[7] NeoMagic用のpatchの当たったXFree86。要するにアクセラレート効かず
[8] http://www.asahi-net.or.jp/%7Ebe9t-udgw/otakara/u_1'st_ta.jpg とか？
[9] 256色より上の色数の色をXAllocColorすると遅すぎるけど、どうしたら良いのだろう？とか悩んだと言うのは、また、別の話である

の画像の明るさを補正するオプションもつけた。このオプションによって、元の画像の明るさを-255〜255まで、(-255 が真っ黒、0 が元のまま、255 が真っ白) 変化させる事が出来るようになった。これで元の絵が背景に適さないとしても、壁紙にする時に自由に明るさを変えられるようになった。

この拡張はかなりイイ感じであった。jpg,gif を support したことにより、Web 上に転がっているほとんどの画像を変換の必要なしに壁紙に出来るようになったのである。また、副作用として、画像の load がかなり速くなった事も僕の幸せを加速させた[10]。

6 将来の野望

現在の所、これだけの機能でもうほとんど満足してしまったのでこれ以上拡張の予定は無いのだが、いろいろ考えてみるとまだまだ拡張の余地はたくさんある。とりあえず箇条書にしてちょっとだけ自分にはっぱをかけてみる。

- 画像の拡大/縮小。Window の大きさに合わせる、等 (geometry 指定)
- display(@ImageMagic) のような Window の中の好きな位置に絵を配置できるように (geometry 指定)
- focus している Window だけでなく、登録された name の Window を root window から検索してたどって行ってランダムに背景を書き換えちゃうのも、良いかも
- jpg,png,gif,xpm だけじゃなくて、もっと色々な format に対応しているともっと良いかも
- file の入力に "http://hoge.com/hoge/hoge.jpg" なんて、許しているとちょっとだけ便利かも

うーん、geometry 指定位は出来るようにしたいかも。それには画像の拡大/縮小の御勉強をしないと、駄目か。むぅ。

7 配布

現在僕の作った壁紙 patch が

`http://delegate.uec.ac.jp:8081/club/mma/%7Etakkun/fun/handling/wallpaper.html`

に置かれている。

8 謝辞

素晴らしい事に気をつかせてくれた oolong に感謝。

参考文献

3X11 man pages
X11R5 Xlib リファレンスマニュアル
X11R5 AthenaWidget リファレンスマニュアル

[10] その後、rxvt という最初から壁紙が貼れる (それも速い)term があることを知ってがっくりするのは、また、別の話である