RCSを利用したらコードの管理がし易くなるかもしんない

fujita@mma.club.uec.ac.jp

1999年11月

概 要

ここ数年コンピュータは爆発的に普及し、またその性能も飛躍的に向上しました。こういった環境の中で、ごく日常的にプログラムを書く人も増えたことと思います[1]。私も時々思い出したようにプログラムを書いては自己満足にひたるのですが、大抵その類のプログラムは書いたら書きっぱなしになっています。10行位のどうでもいいようなプログラムだったらそのままでも構わないでしょう。しかし、レポート用のプログラムや、何らかのプロジェクトのプログラムだったらそうはいきません。「昔のコードの方が正しかったかも」と気付いたときには、既に自分の頭の中には当時のコードは影も形もないのです[2]。

そこでファイルの更新管理にRCS(Revision Control System)を使ってみることにしましょう。もしかしたらより幸せなプログラミングが出来るかもしれません。

1 ファイルの新規チェックイン

最初に任意のファイル “sample.c” があったとします。これをRCSの管理下に置くには、まず新たにRCSが管理するRCSファイルを作る必要があります。このファイルには過去の修正に関する情報[3]が書かれています。RCSファイルはどのディレクトリにあっても良いのですが、必ず “RCS” というディレクトリの下にある必要があります。ただし、RCSファイルはカレントディレクトリにあっても構いません。しかしカレントディレクトリにごちゃごちゃとファイルがあるのはうっとうしいので、カレントディレクトリの下に “RCS” というディレクトリを作ることにします。

```
% mkdir RCS
```

RCSはRCSファイルを探すとき、カレントディレクトリとカレントディレクトリの直下の “RCS” ディレクトリの中を標準で検索します。そのためこうして “RCS” というディレクトリを作っておけば、この中にRCSファイルが格納されることになります。

次にRCSで管理したいファイルから新規にRCSファイルを作ります。このときに使うコマンドがci(チェックイン)コマンドです。ciコマンドは変更したファイルをRCSファイルに統合・格納させるコマンドです。ファイルを新しくチェックインすると、リビジョン番号が1.1のRCSファイルを作成します。RCSファイルの名前は、特に指定がなければ管理したいファイルの名前の末尾に “,v” を追加したものとなります。

```
% ci sample.c
RCS/sample.c,v  <--  sample.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> SAMPLE of RCS          ← ファイルに関する説明文を入力
>> .                      ← ‘‘.’’(ピリオド) で入力終了
initial revision: 1.1
done
```

これで “RCS” ディレクトリの下に “sample.c,v” というRCSファイルが出来ました。

[1] そうなる前から書いてる人の方が多いかもしれない…。
[2] 「オレは自分の書いたプログラムのdiffを全部暗記してるぜ」という人は…いないよね。:p
[3] 修正者とか修正時間とかコメントとか。

2　RCS ファイルからのチェックアウト

さて、ファイルをチェックインすると、元あったファイルはなくなってしまいます。これは ci コマンドによって RCS ファイルの中に組み込まれてしまったからで、ホテルとかの荷物の預け入れと同じ状態です。ファイルを RCS ファイルから取り出すには co(チェックアウト) コマンドを使います。co コマンドは預けた荷物を引き出すコマンドというわけです。

```
% co sample.c
RCS/sample.c,v  -->  sample.c
revision 1.1
done
% ls -l sample.c
-r--r--r--  1 fujita  wheel   38 Nov 16 01:35 sample.c
```

これでファイル sample.c をチェックアウトしたのですが、このファイルは書き込み許可が開いていません。閲覧専用のファイルとしてチェックアウトされたのです。そのため、無理に書き込み許可を与えてファイルを変更し、ci コマンドでチェックインしようとしても出来ません。

```
% chmod u+w sample.c
% emacs -nw sample.c
% ci sample.c
RCS/sample.c,v  <--  sample.c
ci: RCS/sample.c,v: no lock set by fujita
```

書き込み可能なファイルをチェックアウトするには “-l” オプションを付けてリビジョンをロックして排他制御をする必要があります。例えばリビジョン A のファイルを書き直している最中に他の人によって RCS ファイルが更新され、リビジョンが A+1 になったとします。ここで書き直しが終わり、RCS ファイルにチェックインするとリビジョンは A+2 となります。これ以降チェックアウトする最新のリビジョンは A+2 ですが、これはリビジョン A+1 の変更は含んでいません。このような事態を避けるためにリビジョンをロックするのです。

```
% co -l sample.c
RCS/sample.c,v  -->  sample.c
revision 1.1 (locked)
done
% ls -l sample.c
-rw-r--r--  1 fujita  wheel   38 Nov 16 02:07 sample.c
```

リビジョンはロックされ、チェックアウトしてきたファイル sample.c は書き込み許可が開いています。これでファイルを書き直すことが出来ます。

3　再びチェックイン

変更が加えられたファイルを再びチェックインするには ci コマンドを使います。

```
% ci sample.c
RCS/sample.c,v  <--  sample.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> CHANGE REVISION 1.2     ← 新規リビジョンに対するメッセージを入力
>> .
done
```

これで最新のリビジョンが 1.2 になりました。

しかし、これではファイルを更に直したいときに、もう一度 co コマンドでファイルをチェクアウトしなければなりません。この手間を軽減するために、ci コマンドに “-l” オプションを付けてチェックインをすることも出来ます。このオプションはファイルをチェックインし、かつそのリビジョンをロックしておくことを意味します。これによって再びファイルの編集を続行出来ます。

また、リビジョンをロックしないままファイルを更新してしまった場合、もしチェックアウトすれば今までの作業は水泡に帰します。そこでリビジョンを rcs コマンドの “-l” オプションでロックし、それから ci コマンドでチェックインします。

```
% ci sample.c
RCS/sample.c,v  <--  sample.c
ci: RCS/sample.c,v: no lock set by fujita
% rcs -l sample.c
RCS file: RCS/sample.c,v
1.2 locked
done
% ci sample.c
RCS/sample.c,v  <--  sample.c
new revision: 1.3; previous revision: 1.2
enter log message, terminated with single '.' or end of file:
>> CHANGE REVISION 1.3
>> .
done
```

4　リビジョン番号を指定したチェックイン、チェックアウト

ci コマンド、co コマンドはオプションでリビジョン番号を指定することが出来ます。例えば、今リビジョン 1.3 のファイル sample.c を編集して、リビジョン 2.1 としてチェックインしたいとします。これは次のようにします。

```
% ci -r2.1 sample.c
```

これで次回以降のチェックインのリビジョン番号は 2.2,2.3,2.4 のように増えることになります。こうなった場合、リビジョン 1.4 や 1.5 といったリビジョンを作ることは出来ません。

リビジョン 1.2 のファイルをチェックアウトしたいときは、同じように

```
% co -r1.2 sample.c
% co -l1.2 sample.c   ← ロックする場合
```

とします。さらにこのファイルを編集し、チェックインするとリビジョン番号は 1.2.1.1 となります。これはリビジョン 1.2 から枝 1.2.1 が出来て、その枝の一番目であるから 1.2.1.1 がリビジョン番号となります。これ以降のリビジョン番号は 1.2.1.2,1.2.1.3,1.2.1.4 のように増えることになります。

5　リビジョン管理情報の参照

リビジョン情報の一覧を得るには rlog コマンドを使います。下の二つのコマンドは同じ出力を表示します。

```
% rlog sample.c
% rlog RCS/sample.c,v
```

これによってロックの情報等が引き出せます。

また、管理情報をファイル中に埋め込むことも出来ます。マーカと呼ばれる特殊な文字列をファイル中に書いてチェックインすると、チェックアウトするとき RCS がその部分に管理情報を付与してくれます。

```
% cat sample.c
/* $Id$              ← マーカ
*/
main(){
  printf(‘‘Revision 1.1\n’’);
  printf(‘‘Revision 1.2\n’’);
  printf(‘‘Revision 1.3\n’’);
  printf(‘‘Revision 1.4\n’’);
}
% ci sample.c
% co sample.c
% cat sample.c
/* $Id: sample.c,v 1.4 1999/11/15 18:41:44 fujita Exp $  ← マーカが展開される
*/
main(){
  printf(‘‘Revision 1.1\n’’);
  printf(‘‘Revision 1.2\n’’);
  printf(‘‘Revision 1.3\n’’);
  printf(‘‘Revision 1.4\n’’);
}
```

詳しいオプション、動作、マーカに関する情報は以下の文献を参照して下さい。

参考文献

[1] 伊藤和人,「入門 Make&RCS make&RCS による効率的プログラミング技法」, 秀和システム

[2] FreeBSD 日本語マニュアルプロジェクト,「ci(1),co(1),rcs(1),rcsintro(1),rlog(1)」, http://www.jp.freebsd.org/man-jp/

PGP とかポートスキャンとか

nice20

1999 年 11 月 XX 日

概 要

どーも、nice20 です。sanjigen さんからは PGP6.x 関係の話を書けーとのお達しがあったのですが、以前書かれたネタと重複するっぽいのでちょっとだけです。代わりにポートスキャンネタで書きます。でも、私はかなり〜スキル足りない上にわかってないところもかなりあるので間違ってたら「こいつわかってね〜」と笑ってください（あう..）なお、使用した機材はお家マシン（P2-300Mhz − WINDOWS98）とノート（P2-266Mhz − Linux）です。他の環境で動作が違うんじゃぁ〜というひとすみません（ぺこぺこ）、次のページにジャンプしてください。また、本文の内容の正誤などについては一切責任をとりません。くれぐれも鵜呑みにしないようにお願いします。では、それでもいいという人だけどうぞ..

1 ちょっと PGP な話

UNIX 版の PGP は昔と殆ど扱い方は変わってなく暗号化キーの最大ビットに 4096bit まで指定できるようになった位です。（細かいバグフィックスや色々内部で変わっているのはお約束..）メインの使い方は sanjigen さんが昔書いたらしいのでそっちを参考にしてください。（同じ事書いても仕方が無いし..）

で、WINDOWS 版の PGP6.0.1 では PGPDisk という物が追加されています。これは WIN の HDD 上に一定の領域を確保し、それを仮想ディスクとしてマウントして使用するものです。（CD 革命や携速の HDD 版みたいな感じです）WIN システムからはでかいファイルが一つあるように見えます。当然の事ながらそのディスク内容は暗号化されていますので 16 進ダンプして覗いてみたりしてもさっぱりわかりません（笑い）また、復号化（この場合はマウント）にはパスフレーズが必要となります。会社など多人数で使用している WINDOWS マシン上で他人に見られたくないファイルがあるときはこれを使えば幸せになれるかもしれません。

欠点

WIN ファイルシステム上の仮想ディスクなのでデリートには無力。（NT で使えって話も.. 結局は ADMIN には消されるけど...）

他の機能では WIPE というものがあり、これはディスク上からファイルを完全に消去します。ファイルシステムからの参照を削除するのみでなく、ファイルのあった領域をランダムビットで上書きするらしいです。サルベージ屋さん大困りの機能ですね〜

WIN にセキュリティは無い。などと云われていますが（まあ本当にそうなんだけど）端末を直接いじられました、データ取られました、では恥ずかしいので本当に重要なデータは暗号化してみてはいかがですか？

余談

書き終わった後に PGP6.5.1 のインターナショナル版が出ている事に気づきました。なんか新しい機能が追加されたらしいので誰か後で書いてください（笑）

2 ポートスキャンな話

これがメイン（のつもり）です。一般的にポートスキャンは対象のサーバーにおいてそのポートにおけるサービスが利用可能か調べるものですが、概ねアタックの下調べや公開（隠れ？）プロキシサーバーを探す事に使われています。（なんでプロキシなんか使うの？と思う人は多分使う必要はありませんよね〜）最近色々と問題になっていますので、お遊びで他人のサーバーにやる事はお勧めできません。また、やる場合はすべて自分の責任でお願いしますね。

1　実際にやってみる

一番簡単な方法として telnet を使ってみましょう（接続元を your.com、対象を hoge.com とします）

```
your # telnet hoge.com 23
```

レスポンスとしては以下の通りです。

＞ポートは閉じている（接続不可）

```
Trying hoge.com...
telnet: Unable to connect to remote host: Connection refused
```

＞ポートは開いている（接続可能）

```
Trying hoge.com...
Connected to hoge.com.
Escape character is '^]'.
```

また接続されたポートや動いているサービス、ルーターなどにフィルタリングされた場合にはそれぞれ別のメッセージを吐くと思います。

この場合ちゃんとログをとっているマシンなら（Linux-syslogd の場合）

```
Nov  XX 23:05:10 hoge.com sshd[714]: log: Connection from your.com port 2093
Nov  XX 23:05:11 hoge.com sshd[714]: fatal: Did not receive ident string.
Nov  XX 23:06:23 hoge.com in.telnetd[720]: connect from your.com
Nov  XX 23:06:23 hoge.com telnetd[720]: ttloop:  read: Broken pipe
Nov  XX 23:06:23 hoge.com sshd[721]: log: Connection from your.com port 3662
Nov  XX 23:06:23 hoge.com sshd[721]: fatal: Did not receive ident string.
Nov  XX 23:06:24 hoge.com in.ftpd[722]: connect from your.com
Nov  XX 23:06:24 hoge.com ftpd[722]: FTP session closed
```

のようにコネクトされたものはログに残ります。

またいちいち手動でやるのが面倒なひとは、Perl などでスクリプト組むかネットでスキャンプログラムを探してきましょう。

2　仕組み

前述した方法では、指定のポートに接続開始－＞接続－＞切断と、いう手順で行っています。ちょっと tcpdump して見てみましょう your（クライアント）と hoge（サーバー）の通信内容です。

この場合のデータフォーマットは接続元. ポート ¿ 接続先. ポート : フラグ その他データ＆フラグ～となっています。（時間のフィールド、一部内容は削除）

```
your.1035 > hoge.telnet: S 20488139:20488139(0) win 8192 (DF)
hoge.telnet > your.1035: S 3889976118:3889976118(0) ack 20488140 win 32120
your.1035 > hoge.telnet: . ack 1 win 8760 (DF)
your.1035 > hoge.telnet: F 1:1(0) ack 1 win 8760 (DF)
hoge.telnet > your.1035: . ack 2 win 32120 (DF)
hoge.telnet > your.1035: P 1:13(12) ack 2 win 32120 (DF)
your.1035 > hoge.telnet: R 20488141:20488141(0) win 0 (DF)
```

また TCPDUMP における略称は

S : SYN（コネクション確立要求）

P : PUSH（バッファリングせず、即時にデータを送るよう TCP に要求）

F : FIN（コネクション開放要求）

R : RST（コネクション強制切断要求）

. : フラグビット無し

ACK : ACK ビット。次のフィールドが ACK ナンバーである事を示す

WIN : 受信バッファサイズ

DF : フラグメント禁止ビット（ON）

です。知らない人のために一応...

通常、TCP スキャンはコネクションを張る。即ち 3Way ハンドシェークの完結までを試します。よーするにフルコネクトするわけですね。

まず一行目でサーバーの telnet(23) ポートに対してシーケンスナンバー 20488139 で SYN パケットを送っています。

二行目で一行目のシーケンスナンバー+1 を ACK ナンバーとして、下りのコネクション確立のための ACK ビットの立った SYN パケットを返信しています。

三行目でクライアント側がコネクションの確立を行い ACK ナンバーをリセットし 1 から振りなおしています。

この時点で 3Way ハンドシェークが完了し、接続が確立しますのでアプリケーション層でパケットを扱うデーモンなどは接続元の IP を知る事ができます。

3 スキャン色々

じゃあ相手にわからない様にスキャンできないじゃん。と、お思いでしょうがアプリケーション層でロギングしているシステムに対しては 3Way ハンドシェークの完結まで行わなければ IP は通知されないわけです。（他にも方法がありますが..）それでは見つかりにくいスキャン方法をいくつか紹介（tcpdump での出力も一部載せてあります）

1 half-open スキャン

SYN スキャンとも云われます。まずサーバーに対して SYN パケットを投げつけると、ポートが空いているときにはサーバーから SYN フラグと ACK フラグのセットされたパケットが送られてきます。また、閉じているときには RST フラグがセットされたパケットが送られてきます。この時点で切断してしまえばコネクションを張らずにポートが空いているか調べられるわけです。サーバー側にはクライアントからコネクション確立の ACK パケットがこないのでコネクト失敗となりアプリケーションに IP は通知されません。おまけとしてサーバーから送られてくるパケットには OS 特有の WIN 値（受信バッファサイズ）がついてるのでそれから OS の特定も可能です。

```
PORT-CLOSE
your.40039  > hoge.telnet: S 4147027676:4147027676(0) win 3072 # SYN送信
hoge.telnet > your.40039: R 0:0(0) ack 4147027677 win 0 # RST受信

PORT-OPEN
your.33913 > hoge.www: S 3103462978:3103462978(0) win 4096 # SYN送信
hoge.www   > your.33913: S 3678627738:3678627738(0) ack 3103462979 win 32696(DF) # ACK受信
```

2 FIN スキャン

コネクションも張られていないのにいきなり FIN パケット投げつけます。閉じているポートは何投げても RST が戻ってきます。元々コネクション開放要求なので開いているポートからは何もかえってこない仕様のようです。また WINDOWS などは開いていても何故か RST パケットが返ってくるので判別不可です。（UNIX 系では一部不可）

```
PORT-CLOSED
your.59511  > hoge.telnet: F 0:0(0) win 2048 # FIN送信
hoge.telnet > your.59511: R 0:0(0) ack 0 win 0 # RST受信

PORT-OPEN
your.48031  > hoge.www: F 0:0(0) win 1024 # FIN送信
(戻り無し)
```

3 UDP スキャン

UDP の場合コネクション張らないうえにデータが到達したか分からないので確実な方法とはいえませんが、ICMP パケットの戻りはあるので調べる事は可能です。ちなみに TCP コネクションを張るポートは引っかからないので悪しからず...（echo や chargen は探せますね～）

```
PORT-CLOSED
your.8648  > hoge.78: udp 0
hoge       > your: ICMP :hoge udp port 78 unreachable

PORT-OPEN
your.2393  > hoge.echo: udp 0
hoge.echo  > your.2393: udp 0
```

他にも色々あるので興味ある人は自分で探してください。

結局はパケットレベルでロギングすればすべて見つかります。ただ、大規模なシステムではすべてログにとれば膨大なデータ量となりますし、概ねセキュリティにうるさいような場所以外では syslogd に吐かせる程度のログ位しかとってないんでしょうね～

余談

家から PPP 接続していると、まれにスキャンしてくる人もいますが WIN98 マシンをスキャンしてどーするんでしょうか（笑い）ポート番号 30000 万台だったから BO2K ねらいかな？(139

に奇形パケット投げつけるアホもたまに引っかかりますが..）PPP アクセスの IP レンジでもログっている人はいるかもしれないのでむやみにスキャンしない方がいいですね～（WIN でも tcpdump に似たツールがあります）

3　後書き

セキュリティな話とポートスキャンな話でした。ポートスキャンもうまく使えば自分のシステムのセキュリティチェックに役立つはずです。少なくとも IP リーチャブルなマシンはセキュリティに気を使うべきでしょう。（使わないサービスは切っておくに越した事は無いですし..）だからといって他人のサーバーをスキャンしまくって管理者から苦情メールなんかもらっても私は知りませんよ。

では、すべては自己責任で....

ポケステのプログラミングをしてみようハードウェア編

takkun@mma.club.uec.ac.jp

1999 年 11 月

1　ポケステって？

SCEI さんから、PlayStation [1] というゲーム機用のオプションとして発売されている PocketStation は、実は我々一般人がプログラムを組んで、遊ぶ事が出来ます。というのも、秋月さんのキットでも売られているように、PlayStation のメモリーカードは、簡単な自作のキットと適当なソフトウェア (なんと Web で公開されている！) で読み書きが出来てしまうのです。もちろん、同じ方法で件の PocketStation へもデータの読み書きが出来るわけです。PocketStation やメモリーカードへの読み書きをする機械は、簡単に自作出来ます。それに、先人達がいろいろと環境を整備してくれているので、我々素人は全然苦労せずに開発環境を整え、その上 C でプログラムを書いてもおっけーという、素晴らしい事になっています。と、いうことで、今回は、その PocketStation でのソフト開発について、いろいろと書いてみたいと思います。

2　とりあえずハードウェア

僕は 5 インチの FDD につながるのメスコネクタをごりごり削って、PocketStation に刺さるように作りました。こんな感じです。(図 1、図 2 参照)。

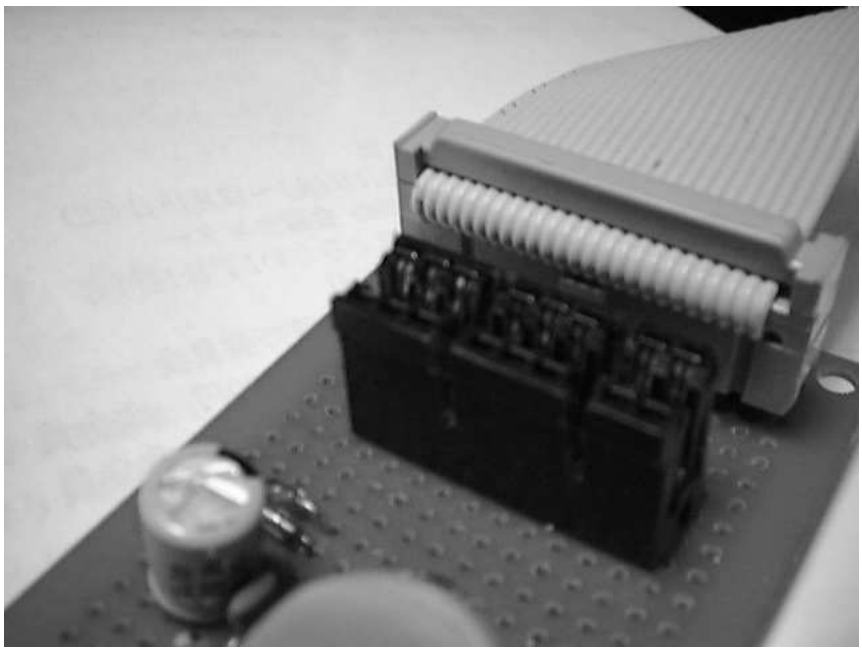


図 1: 削られたコネクタ

[1] 有名ですね。セガのゲームは世界いちぃ！とかいうイベントで叩き壊されたり、ToHeart が移植されたりした、アレです。

図 2: うまいこと刺さる PocketStation

で、その配線なんですが、僕の使っている読み書き用ソフト[2] では、単純に PC のプリンタポートから、そのまま配線して、あとは適当な電源から、安定した電源をぶしっとつないでやればおっけーです。[3] まぁ、ハードウェアについては、これくらいにしておきます。一応、市販のモノもあるらしいので、それをつかうってのも楽で良いかもしれません。

3 とりあえず遊んでみよう

ポケステで開発を始める前に、ハードウェアが適当に動いているかのチェックとか、その前にいろいろ遊んでみようとか、そんなお話が無い分けではないので、既にポケステでソフトを開発している先人たちが作ってくれた面白そうなソフトをいろいろ持って来てつっこんで動かしてみると良いでしょう。URI を末尾にリストしておいたので、適当にいろいろ拾って来て遊んでみましょう。

4 ポケステのスペック

ポケステで開発を始める前に、ポケステのスペックを知っておくと後々便利ですので、スペックを確認しておきましょう。SCEI さんの発表[4] によれば、

ＣＰＵ ＡＲＭ７Ｔ（３２ビットＲＩＳＣプロセッサ）

メモリ

ＳＲＡＭ ２Ｋバイト

[2] メモリーカードキャプターさくら URI:http://ww1.tiki.ne.jp/ nissie/mccs/
[3] と、いっても、僕は面倒なので、推奨されていないプリンタポートから電源供給ってのでなんか、やっちゃってますけれど。
[4] http://www.scei.co.jp/dearscei/pr/981008.html

フラッシュＲＡＭ １２８Ｋバイト（メモリーカード兼用/バッテリ不要）

グラフィックス ３２×３２ドットモノクロ液晶表示

サウンド 超小型スピーカー（１０ビットＰＣＭ）×１個

スイッチ 入力ボタン×５個 リセットボタン×１個

赤外線通信 双方向（ＩｒＤＡ準拠)、リモコン送受光部

ＬＥＤ表示器 １個（赤色）

バッテリ ボタン電池（ＣＲ２０３２）×１個

その他

- カレンダ機能
- 個別ＩＤ（Identification ＝識別番号）

最大外形寸法 ６４×４２×１３．５（縦×横×厚さ）

質量 約３０ｇ（電池含む）

と、いうことです。…だけでは何が何だかわからないので、一応僕の主観に基づいた説明でもしてみます。まず、CPU、32bit です。RISC なんだそうです。ARM[5] とか言ってます。ARMってなんなんでしょう？よくわかりません。あ、どうも、ARMっていうのは組み込み用の小さな効率の良い CPU で、回りのいろんな周辺チップを含めて、一つのチップにしやすいんだそうです。へぇ。凄いですね。次にメモリ、SRAM が 2Kbites らしいです。フラッシュが 128k というのは、PlayStation のメモリカードとしての 15 ブロックというのとなんとなく同じですね。次にグラフィクス、32x32 の白黒二値らしいです。うぅん、狭くて、表現力にちょっと、欠けるようです。せめて中間色の表示できる四階調位にしておいてくれたら良かったのに…まぁ、贅沢を言うのはよしましょう。次、サウンドです。10bitPCM らしいです。ほぅ。PCM ですか。なるほど。てことは、うまいことやれば「みゆっ」[6] とか、言わせる事が出来そうです。うーん、しょぼいグラフィクスと比べて、これは、贅沢ですね。んじゃぁ、次、スイッチ？あぁ、ボタンの事か。ふむ。上下左右と選択ボタンで全部で五個のボタンがありますね。リセットに一つと書いてありますけれど、リセットを押してもリセットされるだけなので、リセットボタンは全然ボタンとしては使えないじゃないですか。ちぇ。うーん、それでも、ボタンが五個ってのは、ナンですね。せめて、選択とキャンセルの二つのボタンがあれば良かったのに。まぁ、贅沢は言わないいわない。んじゃ、次。赤外線通信だそうです。おっ、IrDA とか書いてありますね。すごいや。IrDA 準拠って事は、PC と通信して、なんだかいろいろ出来そうです。ふぅむ。良いですね。あとは…あ、カレンダ機能とかいうのがついてますね。これって…多分、時計の事ですね。ふむ。時計がついているという事は、ゲーム内にこちらの日常の時間を導入できるという事ですね。ふむー。で？後は…個別 ID ですか。ふむ。個別 ID…赤外線でポケステ同士で通信したりする時に、なんか、約に立ちそうです。

ふむ。なるほど。なんだかちょっと非力そうな所がありますが、一応ゲーム機的な最低限の事はそろっていて、いろいろ出来そうです。ちょっと夢が膨らんで来ました。

[5] http://www.arm.com/
[6] EmiClock の起動音として有名ですね

5　開発環境を整えよう

では、とりあえず開発環境を整えようと思います。ポケステの CPU は ARM ですので、ARM のクロスコンパイラを手に入れます。ARM のクロスコンパイラって何処にあるんでしょう？えぇ。簡単です。gcc のクロスコンパイラとして ARM の奴を作ってしまえば良いのです。おっと、さらに簡単な事に、gcc の ARM クロスコンパイラはもう先人達が package にしておいてくれちゃっています。

```
http://www.geocities.co.jp/SiliconValley-PaloAlto/6226/
```

ここです。素晴らしい。僕達は make するだけで ARM クロスコンパイラを手に入れる事が出来るのです。いやー、簡単ですね。あれ？あれれ？ここにはなんと PocketStation のサンプルの source が落ちてますよ？わーい。ついでに、これをもらって来ちゃいましょう。

あれ？make だけでは通らなかった？それに俺は BSD なんて使ってないぞ？あー、そうか。そういう可能性もありますね。そんなあなたのために、DOS 版の gcc の ARM クロスコンパイラを作って下さっている人も居ます。

```
http://plaza4.mbn.or.jp/~iruka/arm.html
```

こちらです。いやー、素晴らしいですね。こっちだと、既にコンパイルまでわれているので、僕等はインストールするだけで良いのです。なんてらくちんなんだろう。先人達に感謝です。

6　手に入れたサンプルコードをコンパイルしてみよう

さて、あなたの gcc クロスコンパイラはちゃんと動いているでしょうか?ちょっと試してみましょう。あ、そうだ。さっき手に入れたサンプルのコードをコンパイルしてみることにしましょう。うーん、テトリスと、ナンバーズ、どっちにしましょうか。とりあえず、source code の小さそうなナンバーズにしましょう。おぉ、親切な事に `Makefile` がついてます。わーい。じゃぁ、`make` しちゃいましょう。[7]

make 出来ました？ほいではポケステに書き込んで…って書き込めませんねぇ。なんででしょ。うーん、困りました。不思議です。んじゃぁ、配布されているナンバーズのバイナリと比べてみましょう。って、あれれ、ファイルサイズからして違うじゃないですか。それに、なんだか先頭からして変なバイナリが追加されているようですよ？それに、この先頭のバイナリ、なんとなくファイル名っぽい感じです。それに、後ろの方はなんだか 0 で埋めてあるみたいです。それに、先頭のバイナリの量を抜くと、ちょうど 8192bytes になるではないですか。ふむー？なんとなく解ってきました。ナンバーズはポケステ上ではメモリブロックを 1、消費します。それっていうのは多分 8192bytes なんでしょう。だとすると、15 ブロックセーブできるポケステのフラッシュの大きさは、`8192bytes*15=122880bytes` です。ふむー？なんとなく`8192bytes*16=131072bytes=1024*128=128Kbytes` てな気がします。ふむふむ。解って来たようです。じゃぁ、その先頭のバイナリをそのまま僕等のコンパイルした奴にもくっつけて、後ろは 0 で埋めてしまいましょう。

こんな感じのプログラムになるでしょうか。

```
#!/usr/local/bin/perl

($name, $from, $to) = @ARGV;

if($to eq ''){
```

[7] あ、クロスコンパイラをインストールしたディレクトリを `TARGET` に指定しないとうまく通らないかもしれないですよん。

```
    print STDERR "usage: obj2pda name from_file to_file\n";
    exit(0);
}

$name = 'TEST_PPP012345' if($name eq '');
die("name letter #7 must 'P'!\n") if(!($name =~ m/^......P/));

die("$to file exists.\n") if( -f $to );
open(OUT, ">$to") || die("cannot create $to\n");

open(IN, "&lt;$from") || die("cannot read $from\n");

select(OUT);

# ファイル名出力
print $name;
$n = length($name);
while($n &lt; 16*3 + 6){
    printf("%c", 0);
    $n++;
}

# 中身出力
$n = sysread(IN, $buf, 0x2000);
die("sysread fail.\n") if($n eq undef);
print $buf;

if($n &lt; 0x2000){
    while($n &lt;= 0x2000){
printf("%c", 0);
$n++;
    }
}
```

まぁ、要するに、適当なファイル名臭い文字列を入れて、その後ろを8192(==0x2000)の大きさまで'0'で埋めるという事をしています。あ、ちょっと待って下さい。一つ言い忘れていました。どうも、先人達の解析情報によると、ファイル名の 7 文字目は必ず'P'である必要があるようです。

で、どうでしょう？書き込めたでしょう？で、実行してみると…動きません。なんだかバグってしまいました。うーん、困りました。何かがおかしいようです。何がおかしいんでしょう…って、ちょっと、待って下さい。gcc の ARM クロスコンパイラには、なんだか ARM 関連のオプションがいろいろあるみたいです。うーん、これらをつけてみるってのは、どうでしょう。ちょっと、読んでみます…

ふむ。なんだか、以下のオプションをつける方がいいような気がします。

`-mcpu=arm7tdmi` ARM には種類があるので、これでターゲットの ARM の種類を指定します。スペックによれば ARM7 らしいので、arm7 に…しようと思ったら、無いのでarm7 では唯一存在するarm7tdmi

を指定しておきます。

`-mthumb-interwork` ARM のマニュアルによれば、ARM には、32bit の ARM mode と、16bit の thumb mode があるらしいです。なんだかポケステの制御にはシステムコールをしたりするらしいので、そのシステムコールを呼んだ先が thumb mode だったりしても大丈夫なように、一応このオプションをつけときます。

`-mapcs-32` 一応、32bit で動いて欲しいので、このオプションをつけます。

`-msoft-float` ポケステには hard-float がついているのか、スペックからではわからないので、一応、このオプションをつけときます。でも、gcc の info によれば、soft-float は ARM の種類によってはちょっと動作がおかしいかもしれないらしいので、libm は使わないで済むなら使わない方が良いような気がします。

全部合わせると、

```
-mcpu=arm7tdmi -mthumb-interwork -mapcs-32 -msoft-float
```

に、なります。これをつけて、もう一度コンパイルしてみたらどうでしょう…おぉ、動きました。素晴らしい！我々はポケステの開発環境を手に入れました！

7 おつぎは来月

と、いうことで、そろそろお時間となりました。来月はサンプルに手に入れたナンバーズのソースを読みながら、ポケステの内部構造に迫ってみたいと思います。お楽しみに！

8 参考 URI

ポケステ開発関係の URI です。参考になれば幸いです。

`http://www.arm.com/` ポケステの CPU は ARM7T らしいです。

PlayStation/PocketStation development with BSD: `http://www.geocities.co.jp/SiliconValley-PaloAlto/` PocketStation(や PlayStation) な gcc-*の package とか。サンプル source が置いてあってしやわせ。

ARM CPU の部屋： `http://plaza4.mbn.or.jp/ iruka/arm.html` Win9*な ARM cross compiler とか、ARM の何処が楽しいのか、とか。

ソニープレステーション PAD/メモリの解析: `http://www.vector.co.jp/soft/data/game/se020797.html` プレステのパッドやメモカについての解析情報。一応メモカへの access 方法とか、format とかがかかれているので読むべき思う。

メモリーカードキャプターさくら: `http://ww1.tiki.ne.jp/ nissie/mccs/` ポケステ対応なメモカアクセスツール。いろんな format に対応している。というか、僕はこれしか使った事無いのでよくわかりません。

F学科中間期末試験 傾向と対策

K.T.Omaedah

1999 年 11 月

概 要

９８年に行われた試験について受験の感想と試験の傾向を探りました。しかし９９年度も同様になるとは限りません。各自で取捨選択をしてください。

1　力学第一、第二 (山崎、松澤)

９９年度の問題を一度見た事がありますが、どうやら去年とは感じが違うような気がしました。違うと言うのは問題の傾向ではなく問題用紙の形式などです。これは推測ですが山崎、松澤の両教官が毎年交替で問題を作っている気がします。しかし前述のとおり傾向は一貫していて教科書の例や例題をそのまま問題にしたようなものも見受けられるので、まず教科書を学習するとよいでしょう。

2　微分積分学第一、第二 (横手)

とにかくノートが命でした。試験は授業ノートからほぼそのまま出題されるのでまずノートの学習が第一でしょう。また授業中に教科書の問題を学習するように指示することもありました。二学期、段々出席が少なくなると突然出席をとることがあったので要注意。

3　線形代数学第一、第二 (三沢)

試験は教科書ノートの持ち込みが可でした。(今年は不明) だから問題の解法を自分なりにノートにまとめて試験に望みました。公式の証明はたぶんないでしょうから解法のテクニックを中心にすると良いかも知れません。たまにレポートを要求しました。

4　物質とエネルギー (野上、中平)

この科目も中平、野上の両教官が問題を毎年交替で作っている気がします。しかし傾向は一致していて、教科書の図や演習問題の類題、基本用語の説明が問われるでしょう。授業の最後の方で教官自らが去年の過去問をくばり、「傾向と対策」なるものをやりました。しかし前述のように両教官が交替で問題を作っている気がしますので余裕があればさらにもう一年さかのぼって過去問を演習するとよいかもしれません。

5　波動と光 (村田、今年はC科)

試験は大変に授業に忠実ですのでまずノートの復習をするとよいでしょう。教科書は「なるほどの波と光 (伊東敏男)」、参考書に「振動 波動 (有山正孝)」でしたが去年は後者のほうが重用されていたようです。実際、二重振り子の問題は後者を題材にしていたような気がします。図書館にもあったので暇があったら一読してみるのもよいでしょう。

6 参照

http://w3.uec.ac.jp:8081/club/mma/ omaeda-/kakomon/

nestの構築記

君島 秀征

1999 年 11 月

概 要

今年、MMA のサーバが mola[1] から nest[2] に交代した。その過程を記す。次のサーバ交代の時に役にたてば本望である。詳しくは [2] に記してあるので、根性のある人はそちらを読んでみると良いかも知れない。

1 mola 構築後の MMA の環境

([1] より続く) 当時の MMA はマシンが足らず、サーバである mola も端末として使っていた。mola の X が不安定だったため、凍りついた X Window の画面を表示しつつサーバとして黙々と稼動し続けることが、しばしばあったものである。サーバとしては安定していて、MMA の (メール, DNS, WWW, etc...) サーバの務めを大禍なく果たしていた。ifconfig コマンドに–a オプションが無い、実行ファイルの置き場所が分散している、などの細かな不満はあったが。

それから数年が経ち、mola よりも速いマシンが gont[3] , alpha[4] , atari[5] と導入され、i486 マシンは駆逐された。また、DHCP を導入するためにネットワークを一部分離し、gont を (choparp router, DHCP サーバ) とした。

2 新サーバ導入の案

端末の役目を終えてからの mola は安定して動き続けたが、OS の BSD/OS 2.1 が 2000 年問題に対応していないという大きな問題があった。マシンパワーが見劣りするようになったこともあり、新サーバを構築することになった。新サーバには、mola の機能を引き継ぐ他に、gont の (choparp router, DHCP サーバ) の機能も任せることになった。

1 月 29 日の部会では、BSD/OS 4.0 で mola に代わる新 beat を作ろうという計画が出た。2 月 19 日の部会では、mola の仕事を alpha にやらせようとか、mola が不安定になればみんな仕事するかな、とかいうような話が出た。

3 mola の不調

4 月 11 日、mola で不思議な現象が起きた。それまで問題なく使えていた perl, awk, inc のバイナリの中身の一部が書き換わるという現象で、ファイルの (72×512)Bytes〜$(80 \times 512 - 1)$Bytes の部分が置き換わったり、置き換わったファイルの mtime が新しくなったりした。原因の特定はできなかったが、OS のバグではないかとも思われた。

これをきっかけに、新サーバへの移行が切実なものとして認識された。

[1] BSD/OS 2.1, Pentium 133MHz を 120MHz で駆動。
[2] OpenBSD 2.5, Pentium II 400MHz
[3] FreeBSD, MMX Pentium 200MHz
[4] Debian GNU/Linux, Alpha 21164A 533MHz
[5] FreeBSD, Celleron 300A, 音楽サーバ, コストパフォーマンス重視

4　新サーバ導入の具体案

4 月 28 日の部会で mola で発生した現象について説明し、新サーバについて具体的な話をした。

1　OS の選定

- BSD/OS は、今でも大学はサイトライセンスの契約をしているのか、していたとしてどのバージョンが使えるのか、等を確認する必要がある。free な OS を使い慣れてしまった今では、使用するのが面倒くさく思われた。
- FreeBSD はブランチが 2.2.x 系列と 3.x 系列の 2 本立てになっていた頃で、安定性や開発の方向に疑問があった。
- Linux は詳しい人がいない。

ということで、NetBSD 1.4 にしようということに決まった。が、softupdates があるからという理由で、後に OpenBSD に方針転換した。

2　RAID について

当初は、RAID を欲しがる向きもあったのだが、RAID を組むと atari 級のマシンが複数台組めてしまうほど金が掛かるため、却下となった。

3　HDD について

新サーバには SCSI HDD を使うことが決まっていたが、同容量の IDE HDD も載せて cron で定時ミラーリングをするという案があった。サーバ機に IDE HDD を載せるのを嫌がる向きがあったため、却下となる。

システムと home を別ディスクにすることが決まった。

5　パーツを買う

5 月 15 日、秋葉ツアーを行いパーツを揃えた。kondoh くんには大いに助けられた。

マザーボード	P2B–F	17,640 円
SCSI I/F	AHA–2940UW	16,800 円
ビデオカード	PCI の Trident チップ	1k 円くらい?
メモリ	128MB DIMM	12,547 円
CPU	Pentium II 400MHz	29,800 円
ケース		14,490 円
HDD	IBM DDRS-34560LVD	20k 円くらい
HDD	IBM DDRS-39130LVD	40k 円くらい

総額 160k 円ほどである。

6　名前をつける

秋葉ツアー後、Anna Miller's 仙川店にて shimiz98 氏の 19 歳の誕生日を祝い、その席にて新サーバの名前を決めた。

MMA のマシンの命名権は構築した人にあるので、みんなの意見を聞きつつ、私が決めさせていただいた。候補としては、“atari” と対になる “hazure”, “suka” などがあったが、負の意味を持つ言葉はやめようということで “nest” になった。他に “ego”, “jam”, “suger” という案もあった。

名前の由来は「企業、AC」[3] である。心地良い巣になるようにという願いもある。ちなみに、tree さんのマシンの命名規則にも match しそうなことは命名してから気づいた。

7　組み立て

組み立ては 5 月 15 日に行ったが、買ってきた SCSI ターミネータが変だったり、AHA–2940UW の BIOS が変で boot しなかったりした。翌日には解決したので、OpenBSD 2.4 をインストールし、ssh をインストールした。Pentium II 400MHz の速さは我々に驚きをもたらした。

8　サーバ機能の移行

ネットワーク上に nest を立てて、サーバ機能を少しずつ移していくという方法を採った。しかし、正常に動作しているサーバから移行するため切迫感がなく、作業の進みは遅かった。

本格稼動する前に OpenBSD 2.5 がリリースされたので、2.4 から 2.5 に入れ替えてから移行に取り掛かった。

1　home の NFS

mola:/export/home を nest:/export/home にコピーし、nest で mountd と nfsd を上げる。次に mola の/home@を消し、nest:/export/home を mola:/home に mount する。個々の端末側でも fstab の変更が必要。

うまくやらないと、mola:/export/home には存在するファイルが nest:/export/home に存在しないということになるので注意する。home を移行してしばらくは mola:/export/home を残しておき、各ユーザがファイルの同期に責任を持つ。

home と smtpd のホストが別なので、nest を落とすと mola の smtpd が各ユーザの.forword を読めなくなりマズい... ような気がするが、あまり気にしなかったな。

2　choparp router, DHCP

choparp router 化し、内向きに DHCP サーバも上げる。引数に与える Ethernet インターフェースと MAC アドレスは、外側のものを指定する。外部から誰かが slogin している時には、choparp 化の作業をしたりルーティングテーブルを消したり arp テーブルを消したりする前に断りを入れた方がよい。

3　Web

mola の apache の設定ファイルをそのまま持ってくれば問題あるまい。問題が起こってもさほど深刻ではない。www の CNAME を mola から nest に移す。

4　メール

サーバ移行の中では最も気を遣う作業。nest で sendmail を上げて、うまく動作しそうなことを確認してから、MMA 宛てのメールを nest に向けてもらう。

メールが nest に届くようになったら、各ユーザは mola で一度 inc して、次に nest で inc してもらうことになる。

5　DNS

mola に名前を引きにくるホストがあるはずなので、nest を primary, mola を secondary にする。個々の端末も、名前を nest に聞くように設定する。とするべきなのだが、nest と mola の両方が primary を名乗るという混沌とした状況のまま今に続いている... のかな。

9　今後

DNS の混沌とした状況は、すぐにでも解決するべきである。また、今では MMA のネットワークを 172.21.139.32/27 として完全に分離できたので、近いうちに NIS+を導入したいと考えている。他に、home を atari の HDD にミラーリングし、安心度を高めることも考えている。

今後、IPv6 グローバルアドレスの取得, サークル棟の 100base 化などが控えており、nest も対応が求められる。これらの変化には nest は十分対応していけると考えている。

mola は約 3 年間、MMA のサーバとして稼働し続けてくれた。nest も同じくらいの期間、サーバとして使われると仮定すると、次のサーバ移行期は 2002 年頃だろうか。次のサーバへ移行する際、この文書が少しでも役にたてば幸いである。

(以下、百萬石 2002 年秋号あたりに続く)

謝辞

意見と助言を与えてくれ、遅々として進まぬ移行作業を暖かく見守ってくださった偉い方々、パーツ買いに付き合ってくれた若い方々、に感謝の意を表します。特に、サーバ機能の移行作業のほとんどを引き受けてくれた中原センセイには感謝の言葉もありません。

参考文献

[1] mola.mma.club.uec.ac.jp.private(172.21.139.43) の構築について, kyogoku, 百萬石 1996 年秋号

[2] Yet Anothor diary, 君島 秀征, http://delegate.uec.ac.jp:8081/club/mma/~sanjigen/index–R.html

[3] http://www.fromsoftware.co.jp/soft/ac/ac/main.htm

Psion Series 5の日本語化ソフトの概要

君島 秀征

1999 年 11 月

概 要

Psion Series 5 の日本語化ソフトは、フリーソフトウェア、噂の域を出ないもの、海外通販 only のもの、店頭販売のパッケージソフトと、私が知っているだけで 4 種類 (実際に入手できるのは 3 種類) もある。以下、それらを列挙してみる。

ちなみに、“EPOC Release 5 C++ SDK” 付属のドキュメントには

> In EPOC, 2-byte UNICODE support is built deep into the system.

という記述があるので、一応、真っ当な日本語化は可能なような気がする。確証はないけど。

KEdit

KEdit[1] は小山氏[2] が開発したフリーソフトウェアで、1997 年 9 月に最初のバージョンが出た。現在は河邊氏[3] が保守[4] している。

Psion の内蔵アプリの大半が複数フォントを扱えることを利用して、(39+1) 種の 1Byte 文字フォントを切り替えながら日本語を表示する仕組み[5] を採っている。恐らく NTT jTEX の日本語化と同じ仕組み。以上の仕組みで、Word, Agenda, Data などで違和感なく日本語が使える。ただし、Sheet は一つのセルに一つのフォントしか使えないため、KEdit による日本語化は実質無理。

日本語化の仕組みが少々特殊なので、Psion の外とのデータ連係が難しいという問題がある。が、それを解決するソフト[6] を河邊氏が開発している。

Series 5mx でも特に問題なく使えた。

管理工学研究所の日本語化

管理工学研究所[7] が Psion Software PLC(現 Symbian[8]) と協力して日本語版を開発すると 1997 年 11 月に発表[9] している。物は既に完成しているという噂を聞くが、発売はされていない。世に出す時は、ZAURUS のようなメジャーな物として売り出したいので、電機メーカー各社でプレゼンして回っているという話を、だいぶ前に聞いたことがある。

個人的には、2000 年 5 月に NTT ドコモが出すという EPOC を載せた PocketBoard[10] に、この日本語化 EPOC が使われるのではないかという気がする。

[1] http://antenna.infocity.co.jp/antenna_series5.htm
[2] http://antenna.infocity.co.jp
[3] http://www.hi-ho.ne.jp/~ktkawabe/
[4] http://www.hi-ho.ne.jp/~ktkawabe/densha.html#KEdit
[5] http://www.hi-ho.ne.jp/~ktkawabe/jiskanfont.txt
[6] http://www.hi-ho.ne.jp/~ktkawabe/densha.html
[7] http://www.kthree.co.jp
[8] http://www.symbian.com
[9] http://www.kthree.co.jp/Psion/press.html
[10] http://www.watch.impress.co.jp/pc/docs/article/990527/epoc.htm

McJapanese/32

McJapanese/32[11] は香港の Onflo[12] が開発したシェアウェアで、1998 年 10 月に発表された。同様の McChinese/32 というソフトも存在する。

文字コードには Shift JIS を用いているとのことだが、詳しいことは不明。Word と Agenda では KEdit と同様に muiti–font を用い、Data については内部 2Bytes のアプリを新しく開発した、という話もある。

Series 5mx でも使えるかどうかは不明。

Service Pack 1 が配布されている。

UniFEP for EPOC

UniFEP for EPOC[13] は株式会社エヌフォー[14] が開発したパッケージソフトウェアで、1999 年 7 月に発売された。

文字コードには 2Bytes Unicode を用いている。KEdit と違い、Sheet も日本語化できる。また、KEdit を使って作成したデータを UniFEP 形式にコンバートするツールも付いている。

Series 5mx では、Ver. 1.0 は特に問題なく使えたが、Ver. 1.1 は動作しなかった。

1999 年 9 月に、Ver. 1.1 アップデータが出た。

1999 年 10 月に、Series 5mx に正式対応した UniFEP for 5mx が出た。

[11] http://www.onflo.com.hk/shareware/psion/mcjpn/mcjpn32.html
[12] http://www.onflo.com.hk
[13] http://www.enfour.co.jp/psion/japanese/unifep.html
[14] http://www.enfour.co.jp/inc.html

PicoBSD のおはなし

ottan

1999 年 11 月

1 まえふり

ここんとこ、 MMA の一部で SETI@home[1] が流行っています。
このプロジェクトは RC5 Crack と同様、世界中の遊んでる CPU を動員して電波望遠鏡から得られる膨大なデータをスキャンして、地球外文明の痕跡を探そうと言うもの。しかし、まぁ、それは大義名分で、要するに、自分もしくは MMA を含めての自己顕示欲を満足させたいだけという話もあります (^^;

そこで、自分の管理下にあるマシンを動員するわけですが、昨今では、安心して計算を任せられる Workstation よりも、Windows 端末のほうが速かったりします。
だけど、見かけは綺麗だけど遅いスクリーンセーバーや不安定な DOS 窓じゃ嫌だ、という訳で、one floppy based bsd である PicoBSD で、Windows 環境に影響を与えない、安定した、SETI@home を実行する環境をつくってみることにします。

あと、ここで作ったサンプルは、
`http://delegate.uec.ac.jp:8081/club/mma/%7Eottan/picobsd/`
に置く予定です。

2 PicoBSD ってなに？

PicoBSD は、フロッピー 1 枚で立ち上がる FreeBSD です。

もともとポーランドの Andrzej Bialecki さんが作成されたようです。それが 1998 年 8 月に FreeBSD の CVS に取り込まれ、以後 FreeBSD の一部としてリリースされています。

ダイアルアップアクセス、ルータ、ネットワーク、ダイアルインサーバの 4 つの目的別のセットがデフォルトで用意されていて、それぞれ、最低限のコマンド、セットによっては natd や ipfw などのファイヤーウォール関係やリモート管理用の snmpd やリモートログイン用の telnetd などが含まれています。

これらの既成のセットはディスクイメージも用意されていますので、dd してブートするだけで、ルータ等として機能させることができますし、ダイアルアップバージョンのイメージには ssh まで[1] 入ってたりします。

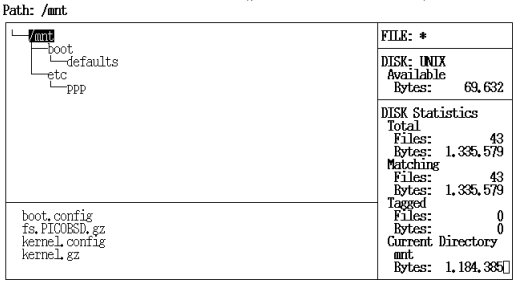
2.2 系列じゃなきゃやだって人には、PicoBSD を 2.2.5 で動かしてる人もいるようですが、和製で、2.2.8 ベースのルータキット [5] [2] をつくっている人もいます。

3 PicoBSD をつかむ

PicoBSD はフロッピーからブートはしますが、動作は完全に MFS 上で行われます。メモリが少なかったころのように、フロッピーがすなわちファイルシステムの全てだったりはしません。ましてや、init がシェルスクリプトだったりもしません (^^;

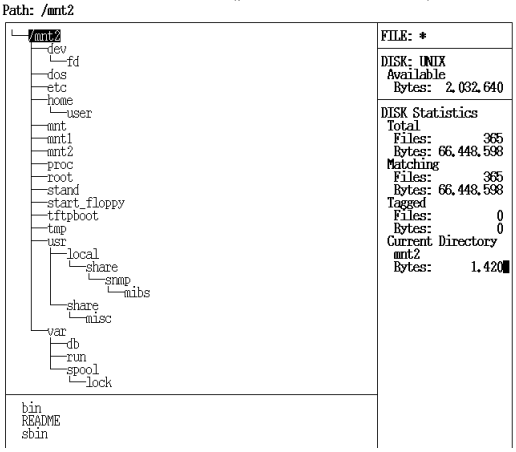
[1] ソースの方では ssh はコメントアウトされてますので、デフォルトの設定でビルドしても ssh は入ってません
[2] OTP(S/Key) に対応してたり、ifconfig で mac address が使えたりと、小技がいろいろつかえるようです

フロッピーの内容 (/mnt に mount) はこんな感じで、



ブートローダーと kernel と /etc はそのまんま、fs.PICOBSD.gz には MFS のイメージが入ってます。

MFS の方の内容 (/mnt2 に mount) は、network セットだとこんな感じです。



実行ファイルは /stand にあり、すべて同一バイナリへのハードリンクとなっています。それ以外のディレクトリはプログラムを動かすのに必要な最低限のファイルやデバイスファイルファイルなどが入っています。あとの、mnt*,start_floppy,dos などは単なるマウントポイントです。

実行ファイルがすべて同一バイナリへのハードリンクというのは、PicoBSD では crunchgen(1) を使って実現されています。man から引用すると、

> クランチバイナリ (`crunched binary`) は、たくさんの別々のプログラムをひとつにまとめて単一の実行形式にしたプログラムです。クランチバイナリの `main()` 関数は、`argv[0]` の値をみて、どのコンポーネントプログラムが実行されるべきかを決定します。複数プログラムをクランチしてひとつにまとめる主たる理由は、インストールフロッピあるいはシステム回復フロッピ上に、できるだけ多くのプログラムを収納するためです。

とあります。シェアードライブラリを動作させるために必要な諸々のものの容量すら削ろうということでしょうか。

あと、プログラムの幾つかは TinyWare というリソースをあまり要求しないもので代用されています。strip された kvm も組み込まれてない kernel で動く ps や netstat とか、getty/login/shell の機能までこなしてしまう init とか、fork しない簡易 httpd なんてのもあります。

4　とりあえず PicoBSD をうごかしてみる

1　準備

まず、開発用のホストとして FreeBSD 3.* のインストールされたマシンと、フルソースを用意しましょう。sys と release の他は組み込みたいコマンドのソースがあれば良いのですが、フルソースと言っても 215 MB 程度ですから、ディスクじゃぶじゃぶなご時世 全部入れちゃいましょう。PicoBSD は src/release/picobsd [3] の下にあります。
これ以降の相対パスはこのディレクトリからのものとします。

次に vnode ドライバが kernel に含まれてなければ、フロッピーのディスクイメージを作成する際に必要になるので、

```
pseudo-device vn
```

をコンフィグファイルに追加して再構築しましょう。

あと、これが一番重要ですが、SETI を実際に実行させるマシンをかき集めてきましょう。
PicoBSD 自体は、8MB 以上のメモリの載った i386 アーキテクチャのマシンであればどこでも動きますが、setiathome が 16M 近くメモリを消費することがありますので、出来れば 32MB 以上のメモリと、自己満足できる程度には浮動小数点演算の速いマシンを用意しましょう。

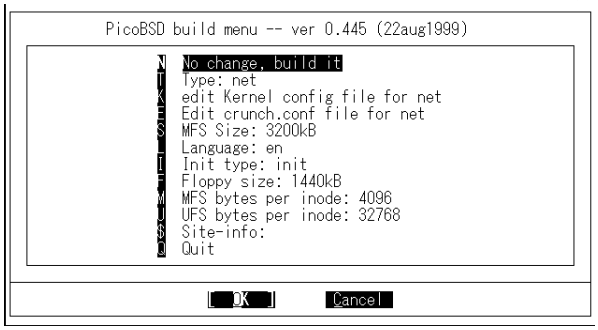
2　build

とりあえず、 network セットを作ってみて、流れをみてみましょう。

まず、 build ディレクトリの下で、

```
#./build net
```

としてやると、コンフィグのメニューが出てきます。



コマンドやドライバを追加する程度の変更ならここで出来ます。kernel config や crunch.conf の edit を選択すると、エディタが立ち上がりそれぞれのコンフィグファイルを変更できます。あとは、language は english か polish しか選べないのと、init は容量や機能からみて oinit に変更することはないので、とりあえず、今回は関係ないでしょう。

適宜変更後、“No change, build it” で作り始めます。できあがったら、 dd するデバイスを聞いてくるので、/dev/fd0 に書き込むとき以外はデバイスファイルを与えてあげましょう。とりあえず、ディスクに書く必要がなければ、キャンセルしてやればいいでしょう。

メニューを抜けると、picobsd.bin という名前で、イメージファイルもできています。あとは適当なマシンでブートすれば ok です。いろいろ試してみてください。

fixit などよりカスタマイズも簡単でそれなりに使えるので結構使えます。

[3] PicoBSD のソースツリーの中の大まかな構成は、トップディレクトリにある README.luigui にちょっと説明があります。また、PicoBSD の etc 下の内容のオリジナルの FreeBSD との違いが表にまとめてられたりもします。

5 PicoBSD で索敵してみる

じっさいに setiathome を動かすためのセットを作ってみましょう。締め切りすぎちゃってるので、ちょっと駆け足ですが、

クライアントには手をつけないで tftp server を用意し、そこに setiathome 用のデータを置くことにします。あと、 mount_msdos して FAT 上に置いといてもいいかも知れません。

とりあえず、追加する機能は、

- dhcp で ip アドレスを取得
- tftp server から get & put
  setiathome のデータファイル、シェアードライブラリ関係
- setiathome を実行

あたりでしょうか。

setiathome はバイナリ配布なので結局これのためにシェアードメモリ関係のファイルが必要です。ld-elf.so.hints,ld-elf.so.1,libc.so.3,libm.so.2 の 4 つのファイルがあれば動きます。ヒントファイルは ldconfig を入れたセットで一度だけ生成しておけばあとは ldconfig は不要です。

では、比較的素直な構成の network セットを元に crunch.conf や /etc の下を書き換えて custom セットをつくっていくので、network セット用のディレクトリをコピーします。`#cp -R net custom`

各セット用のディレクトリの内容はこんなかんじです。

Makefile.mfs MFS イメージ作成用の makefile

PICOBSD カーネルコンフィグファイル

crunch1/ クランチバイナリ作成用ディレクトリ

floppy.tree/ フロッピーの ufs 上にそのまま置かれるファイル群をおさめるディレクトリ

lang/ 各 langage 用のファイル

mfs.mtree mtree 形式の MFS のディレクトリ構成

1 tftp server

ここでは、トップディレクトリに setiathome の実行バイナリやシェアードメモリ関係のファイル、ip adress なサブディレクトリの下に、setiathome 用のデータファイル (*.txt) が置いてあることを想定しています。

2 crunch.conf

まず、クランチバイナリに必要なものを追加します。

custom/crunch1/crunch.conf の srcdirs フィールドに追加したいプログラムのソースの入ったディレクトリのあるパス (要するに /usr/src/bin とか /usr/src/usr.sbin とか) を、progs フィールドに追加したいプログラムを書きます。

一般的なソースディレクトリは既に登録されているので、dhcpclient,tftp,tee あたりを progs フィールドに追加して、つかわなそうなコマンドを削除すればいいでしょう。

ただ、 FreeBSD のソースツリーに外部から取り込まれたプログラム (src/contrib や src/gnu など) は、多少 Makefile を書き換えないとうまく行かない場合があります。

3 PICOBSD

kernel config ですが、dhcp 用に bpf を追加しましょう。デバドラは適当に設定してください。
あとは、ブート時に止まらないよう options INTRO_USERCONFIG を切るぐらいでしょうか。

4 Makefile.mfs

/dev/bpf0 を作成するために、MY_DEVS に bpf0 を追加します。
dhcpclient 用に`cp /sbin/dhclient-script \$\{DESTDIR\}/stand/` を、適当なところに追加します。

5 mfs.mtree

setiathome 用の作業ディレクトリ /seti を、これも適当なところに追加します。

6 floppy.tree

/etc の下に setiathome 起動用のスクリプトを仕込みます。
こんな感じのスクリプトを置くのと、rc の最後に rc.local を起動するよう付け加えましょう。
rc.local

```
TFTP_SERVER=""
/stand/dhclient
if [ "$TFTP_SERVER" = "" ]; then
    TFTP_SERVER=‘/etc/getgw.sh‘
fi
IP_ADDR=‘/etc/getip.sh‘
export TFTP_SERVER IP_ADDR
trap "/etc/tftp_put.sh;reboot" 2
echo "${TFTP_SERVER} ${IP_ADDR}"
/etc/tftp_get.sh
echo ’’
echo ’’
echo ’+----------- PicoBSD @VER@ (SETI) -------------+’
echo ’|                                              |’
echo ’|    Hit  "Ctrl + C"  key,                     |’
echo ’|            If you wont to reboot.            |’
echo ’|                                              |’
echo ’|    Now Running SETI@home on this host.       |’
echo ’|                        output is ttyv1.      |’
echo ’|                                              |’
echo ’+----------------------------------------------+’
cd /seti
setiathome 2>&1 > /dev/ttyv1
/etc/tftp_put.sh
```

tftp_get.sh

```
#!/bin/sh
cd /seti
echo "connect ${TFTP_SERVER}
bin
get libc.so.3 /usr/lib/libc.so.3
get libm.so.2 /usr/lib/libm.so.2
get ld-elf.so.1 /stand/ld-elf.so.1
get ld-elf.so.hints /var/run/ld-elf.so.hints
get setiathome /stand/setiathome
get ${IP_ADDR}/outfile.txt
get ${IP_ADDR}/result_header.txt
```

```
get ${IP_ADDR}/state.txt
get ${IP_ADDR}/temp.txt
get ${IP_ADDR}/user_info.txt
get ${IP_ADDR}/version.txt
get ${IP_ADDR}/work_unit.txt
quit"|tee |tftp
cd /stand
chmod 755 setiathome ld-elf.so.1
```

tftp_put.sh

```
#!/bin/sh
echo ‘‘connect ${TFTP_SERVER}
put /seti/outfile.txt /seti/result_header.txt /seti/state.txt /seti/temp.txt /seti/user_info.txt /se
```

getgw.sh

```
#!/bin/sh
IFS=’’$IFS;’’
(   while : ; do
        read a b c
        [ ‘‘$b’’ = ‘‘dhcp-server-identifier’’ ] \&\& echo $c
        [ ‘‘$a’’ = ‘‘’’ ] \&\& break
    done
) < /var/db/dhclient.leases
```

getip.sh

```
#!/bin/sh
IFS=’’$IFS;’’
(   while : ; do
        read a b c
        [ ‘‘$a’’ = ‘‘fixed-address’’ ] \&\& echo $b
        [ ‘‘$a’’ = ‘‘’’ ] \&\& break
    done
) < /var/db/dhclient.leases
```

参考文献

[1] The Search for Extraterrestrial Intelligence at Home,
http://setiathome.ssl.berkeley.edu/home.html

[2] Small FreeBSD Home Page,
http://www.freebsd.org/%7Epicobsd/

[3] PicoBSD, the small BSD,
http://www.freebsd.org/%7Epicobsd/picobsd.html

[4] TinyWare,
http://www.freebsd.org/%7Epicobsd/tinyware

[5] zensyo@ann.tama.kawasaki.jp,
FreeBSD IP ルーターコンパイルキット,
ftp://ftp.ann.tama.kawasaki.jp/pub/FreeBSD/router

コカコーラの秘密

君島 秀征

1999 年 11 月

概 要

人はなぜコカコーラを飲むのか。そして、なぜマクドナルドのハンバーガーを食べるのか。疑問に思ったことはないだろうか。

実は理由があったのだ。ここで、その謎を明かしていこうと思う。

かつて地球は火星人の侵略を受けたことがあった。H.G. ウェルズの時代のことである。進んだ科学力を持つ火星人の前に人類の為す術はなく、多くの同胞が倒れていった。

しかし、火星人の侵略も終焉を迎える。地球の病原菌に抵抗力を持たない火星人は、地球人の肉をレアで食べた結果、食中毒を起こし次々と自滅していったのだ。

その後、火星人は地球人の肉を食べる安全な方法を考え出したかも知れない。例えば、肉は食べる前に火を通す、など。ということは、火星人の再侵略は大いにありうることだ。地球人は身を守る手段を新たに考え出さなければならないのだ。

“ビジター” において、人類は侵略者に対抗する手段として “赤い粉” を開発した。赤い粉は強力で、これを含む空気を吸ったエイリアンは死に到る。しかも地球人には (ほとんど) 無害だ。しかし、赤い粉の効果を無効化する薬の存在がエイリアンに知られてしまった。赤い粉はもうだめだ。[1]

赤い粉とは別なアプローチもある。地球人の体を、異星人に対して毒となるよう改質する方法である。具体的には、異星人にとって有害な物質を地球人の体内に蓄積すればよい。そう、そのための手段がコカコーラであり、実は人類が侵略者から身を守る術だったのだ。

追記

アメリカでは牛もエイリアンに狙われているらしいので、牛にもコカコーラを飲ませた方が良いと思う。

[1] そもそも、“ビジター” は TV ドラマでありフィクションである。