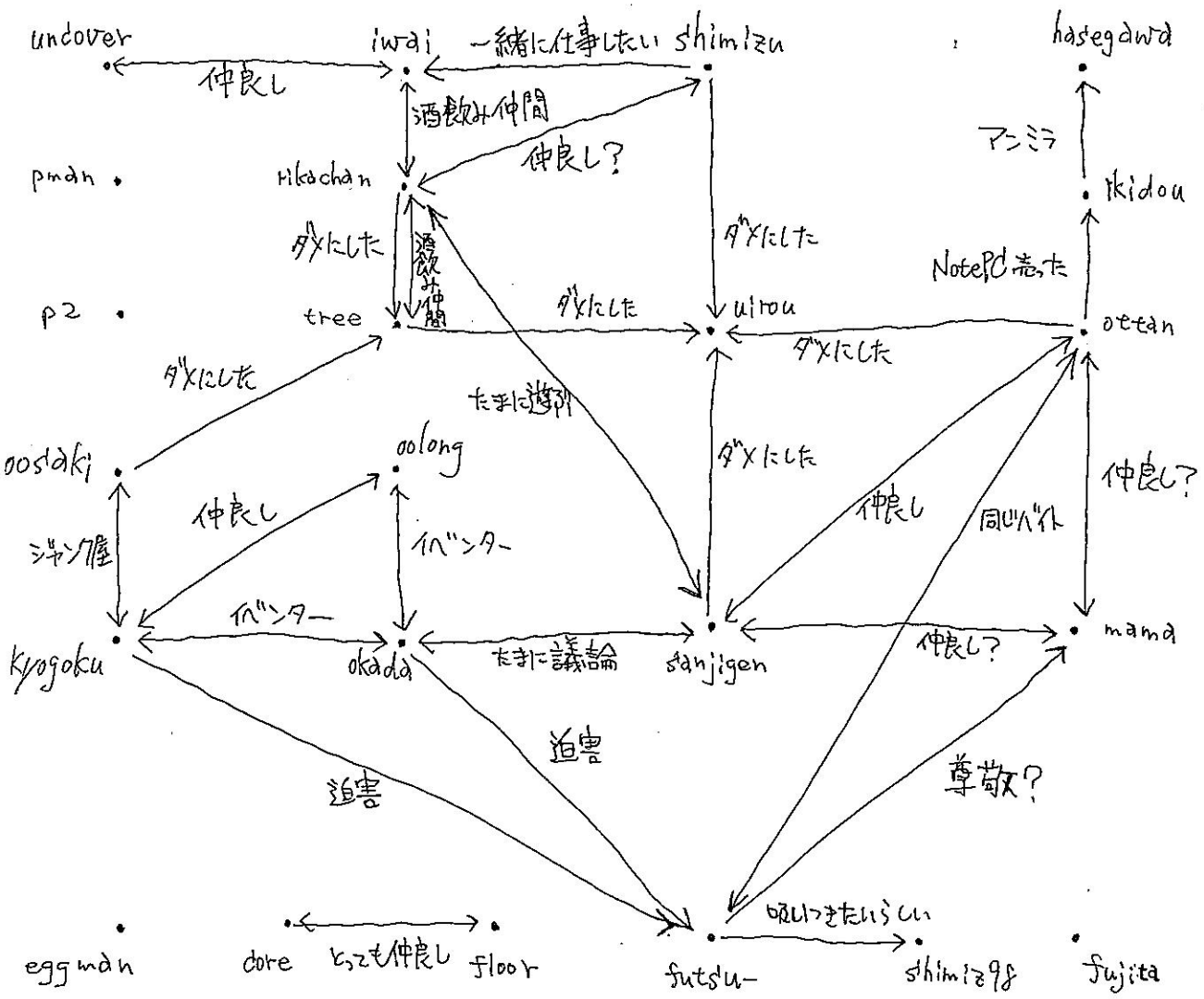

MMA 人物相関図

百萬石

'99年 新歓

新入生歓迎の辞

fujita@mma.club.uec.ac.jp

新入生の皆さん、御入学おめでとうございます。我が MMA は 1976 年に創設され、Micro computer Making Association の文字通り、コンピュータの製作を主とした活動を行ってきましたが、ここ十数年来の半導体技術のすさまじい向上によって安価な高性能マシンが手に入るようになり、その活動もハードウェアからソフトウェアへと対象を移行するようになりました。現在では、UNIX 互換の OS を用いてのソフト開発やネットワーク管理がメインの活動になっています。この冊子を御覧になると、その活動の一端が理解いただけると思います。

さて、ここで MMA の過去の大いなる業績を語れるほど私は MMA 史に通じていませんし、かつ新入生の方々に精神訓話を垂れるほど人間性が確立しているはずもありませんので、

「MMA に入部すると楽しいよ」

ということで歓迎の言葉とさせていただきます。

では充実した学生生活を...

移植性の高いプログラミング
～約束を守るために～

楯岡孝道
(tate@spa.is.uec.ac.jp)

概 要

UNIX 上でプログラミングをする場合、異なる UNIX 間の細かな差異によってせっかくのプログラムが動作しなくなるという問題がある。しかしながら、プログラミング時の多少の工夫によって移植性を向上させ、この問題を減少させる事ができる。本文章では移植性向上の為のヒントを延べ、またその応用として autoconf の使い方の基礎を延べる。

1 はじめに

UNIX Operation System という区分は非常に広く、1969 年の最初の UNIX から最新の UNIX まで様々である。UNIX 上でプログラミングをする場合、異なる UNIX 間の細かな差異によってせっかくのプログラムが動作しない問題が生じる。

4.2BSD などのあまりに古いバージョンの UNIX はともかく、現在でも 4.4BSD UNIX, SUN Solris, SGI IRIX, Linux などが広く利用され、それぞれの間の差異が問題になる。また、さらに古い SunOS, NEWS-OS もいまだに利用されており、POSIX 互換環境という意味では Win32 API や BeOS、MacOS X も含まれるため、差異は更に大きくなる。

しかしながら、プログラミング時の多少の工夫によって移植性を向上させ、この問題を減少させる事ができる。

本文章では具体的な差異を例示し、移植性向上の為のヒントを延べた後に、その応用として autoconf の使い方の基礎を延べる。

なお、本文章では簡単のために C 言語に限定するが、基本的にはどの言語でも同じである。[1]

2 UNIX 間の差異の例

各 UNIX 間の差異には以下のようのものがある。

1. アプリケーションプログラムの名前や動作やパス名が異なる
2. ライブラリ名が異なる、もしくは存在しない
3. ヘッダファイルの名前や必要とされるヘッダファイルが異なる
4. 関数名が異なる、もしくは存在しない
5. 特定の構造体や変数の型が異なる、もしくは存在しない

1 の例としては、`install` コマンドがある。例えば Solaris 2.5 の `/usr/sbin/install` には、FreeBSD の `/usr/bin/install` に存在する -d オプションが存在しない。また、どちらに IRIX 6.3 の `/sbin/install` にある-new オプションは存在しない。

2 の例としては、RPC ライブラリがある。`rpcgen` コマンドで生成した RPC スタブに必要な RPC ライブラリは BSD/OS では `/usr/lib/librpc.a` だが、FreeBSD では `/usr/lib/librpcsvc.a` であり、RedHat Linux では /usr/lib/libc.a に含まれるために明示的にリンクする必要がない。

3 の例としては、termcap ライブラリのヘッダがある。termcap ライブラリの関数は通常 `/usr/include/termcap.h` で定義されるが、Solaris 2.5 では curses ライブラリに吸収されているため `/usr/include/curses.h` で定義される。

[1] もちろん素直に書くだけで可搬である Java などは除く

4 の例としては、cfmakeraw() がある。FreeBSD などの 4.4BSD 系の UNIX には cfmakeraw() が存在するが、IRIX 6.3 などにはこれに相当する関数は存在しない。

5 の例としては、off_t がある。殆どの 4.4BSD 系 UNIX では ssize_t は quad_t (int64_t) であるが、Solaris 2.6 では long であるし、古い UNIX には off_t 自体が存在しない。

3 移植性を向上させる方法

前節のような環境の差異を吸収するには、#ifdef 条件文などを用いて環境に応じて用いる関数やファイルを切り替える。利用している関数が存在せず、代替関数がある場合には、それを使うか、自ら代替関数を作成してそれを使ことにより、同じソースを複数の環境で利用することが可能となる。

その際にも、本節で延べるような工夫が効果的である。

3.1 一般的な関数を利用する

近年では互換性の向上のために、同じことを実行できる複数の関数が用意されている事も多い。例えば、BSD 系 UNIX ではメモリコピーの為に bcopy(3) を用意しているが、現在の 4.4BSD 系では ISO-C/POSIX で標準化された memcpy(3) も用意されている。

このような場合、最初から POSIX などで標準化された関数を使う事により、移植を容易にする事ができる。また、関数によっては POSIX での規格から拡張されているものがあるが、このような拡張部分を使う際には最初に拡張されていない環境の事を考えながら実装することにより、後々の移植の労力を激減できる。例えば、ISO-C の memcpy(3) ではコピーする領域の重複を許さないが、4.4BSD では重複を許すように拡張されている。

もちろん拡張された機能を用いる事により、性能を向上させる手法もあるが、その場合には必ずそれを意識して、重複を許さない場合のコードを追加しておくべきである。

3.2 使用する関数を選択する

ISO-C や POSIX などの規格は一般に「最大公約数」的な性格を持つため、高度な制御を行なう際には不足することがままある。また、古い UNIX の中には規格で定められた関数が不足していたり、その動作が規格に合致していなかったりするものもある。[2]

そのような場合には、環境依存の関数を使い分ける事により対処する。この際に注意すべきこととして、

```
#ifdef __FreeBSD__
```

などの特定の OS に依存した切り分けは極力避け、

```
#ifdef HAVE_CFMAKERAW
```

などと、その機能や関数そのものの存在を示す記述にすべきことである。このように記述することにより、FreeBSD 以外でも cfmakeraw() がある環境であれば、コンパイル時に -DHAVE_CFMAKERAW を指定するだけで、その部分のコードが利用可能になるためである。

3.3 プログラム構造を分割する

環境によっては、特定のことがらの処理を行なう実装が大きく異なってしまう事がある。

例えば、ネットワーク上の通信をモニタする機構としては SunOS の NIT (Network Interface Tap) や BSD 系の BPF (Berkeley Packet Filter) などが広く使われいるが、それぞれに互換性がない。

このような場合には、特定処理を一つの機能モジュールとして分離し、同一の呼び出しインターフェイスを持ち、内部で利用する機構が異なる実装を複数用意する。このようにしておくと、環境に応じて結合するモジュールを切り替える事により、環境の変化に適応することができる。

[2] NEWS-OS 4.x など

4 autoconf

これまでに延べたように、複数の環境でコンパイルでき、かつ正常動作するプログラムを作成するには、コンパイル時にどのようなコンパイル環境であるかを判断し、適切な条件コンパイルを行なわなければならない。

初期のフリーソフトウェアなどでは、この環境の確認をコンパイルするユーザが明示的に行ない、プログラムに指示していた。これは Makefile や config.h の書き換えなどで指示される他、後には対話型のスクリプトによって指示する方法が生まれた。

しかし、ユーザによる指示は誤りを生み易く、またプログラムが高度になるに従って、必要な条件も増加し、ユーザの負担となった。

そこで、これらの環境情報を自動的にプログラムに指示する機構として、imake や autoconf が生まれた。ここでは autoconf について簡単に延べる。

4.1 autoconf の基礎

autoconf はコンパイル/実行環境のチェックを行なうスクリプトを自動生成するプログラム群である。ユーザの立場から言えば、configure スクリプトを作成するプログラムであるといえばわかり易いだろう。

autoconf は様々な機能を持つが、基本的には m4 マクロ言語で記述された configure.in というファイルを読み込み、configure スクリプトを作成する。configure スクリプトはコンパイル時にユーザによって実行され、コンパイル環境を自動確認し、Makefile.in からコンパイル環境に合わせた Makefile を生成する。

configure スクリプトは Makefile.in 中の変数を置換することにより、Makefile を生成する。置換される変数名は`@CC@`, `@DEFS@`, `@LIBS@` など様々である。

例えば @DEFS@ には `-DHAVE_CFMAKERAW` などの環境依存情報が、C プリプロセッサへの引数の形で含まれる。

4.2 configure.in の作り方

autoconf を実行する際にキモになるのは configure.in である。初心者は以下に延べるステップでこれを作成すると良いだろう。

1. autoscan を実行し、configure.scan を生成する
2. configure.in を編集し、autoscan を実行する
3. Makefile.in を作成する
4. configure を実行し、生成された Makefile を確認する

4.2.1 autoscan の実行

autoscan はカレントディレクトリ以下の C プログラムをスキャンし、良く知られている環境依存な記述を configure.scan として出力する。scanfigure.scan はそのまま configure.in として利用できるので、ここで生成した configure.scan を configure.in にコピーし、configure.in のテンプレートとする。

4.2.2 configure.in の編集

configure.in の基本的なマクロは以下の通りである。

dnl コメント

AC_INPUT 元となるソースファイル名が記述され、configure 実行時にはこのソースが srcdir に存在する事が確認される。

AC_OUTPUT 結果が出力されるファイル名が記述される。ここで延べる用途では
`AC_OUTPUT(Makefile)` で十分である。

AC_CHECK_HEADERS 引数で示したヘッダの存在を確認する。存在する場合には
`@CFLAGS@` に
`-DHAVE_TERMAP_H` のような形で出力される。

AC_CHECK_FUNCS 引数で示した関数の存在を確認する。存在する場合には `@CFLAGS@` に `-DHAVE_CFMAKERAW` のような形で出力される。

この他にも多数のマクロが存在するが、これだけでも様々なチェックが可能である。詳細は autoconf の info [5] を参照して欲しい。

autoconf を実行すると、configure.in が読み込まれ、configure スクリプトが生成される。この際、install-sh が存在しないというエラーが出ることがあるが、出た場合には素直にどこかのソースから適当な install-sh スクリプトをコピーしてくるのが良い。GNU のソースや X11 のソースなどに含まれるので、プログラムに合った Copyright のものを利用するのが良い。

4.2.3 Makefile.in の作成

次に生成される Makefile の元となる Makefile.in を作成する。これは通常の Makefile と同じであるが、"@" で挟まれた文字列が configure スクリプトによって置換される。

置換される文字列としては先に延べた`@CC@`, `@DEFS@`, `@LIBS@` などの他にも`@exec_prefix@` や `@INSTALL_PROGRAM@` などがある。これだけで様々な適応が可能である。詳細は info の Substitutions in Makefiles を参照して欲しい。

4.2.4 configure の実行

configure と Makefile.in が用意できたら、autoconf によって生成された configure スクリプトを実行してみる。エラーが起きなければ Makefile.in から Makefile が生成される筈である。

あとは、適宜 configure.in や Makefile.in や元のソースファイルを書き換えれば良い。

幸いにも MMA 部室には複数の UNIX 環境が稼働してるので、各環境で configure と make を実行してみて、正しく動作することを確認すると、リリース前のチェックとしては最高であろう。

5 おわりに

私は機種依存なプログラムが嫌いだ。MZ シリーズを使っていて PC-8001 シリーズからの移植に苦労したとか、J3100ss001 を使っていて、puts にすら機種依存コードを入れる Borland-C に激怒したとか、バイト先で管理したのが NEWS-OS 4.x だったとか、そういう過去のせいかもしれない。

だから、NetBSD 専用とか、Linux 専用とかつまらない事言わずに、みんなで幸せになろう。

6 謝辞

本文章は、私が敬愛してやまない萩野 "いとぢゅん" 純一郎氏の "autoconf ぼくらの約束" [1] の内容と、Kazuya 'Sharl' Masuda 氏の "初心者への GNU autoconf のススメ" [3] の内容を受け、楯岡の立場から加筆、再構成したものです。

私に autoconf を教えて下さった彼らに感謝します。

参考文献

[1] 萩野 "いとぢゅん" 純一郎, "autoconf ぼくらの約束,"
`http://www.itojun.org/hack/autoconf/promise.html`

[2] 萩野 "いとぢゅん" 純一郎, "はっかーずらぱらいす,"
`http://www.itojun.org/hack/`

[3] Kazuya 'Sharl' Masuda, "初心者への GNU autoconf のススメ,"
`http://www.ufo.co.jp/%7Esharl/autoconf.html`

[4] Donald Lweine, "POSIX Programmer's Guide," O'Reilly & Associates, 1991.
`http://www.oreilly.com/catalog/posix/noframes.html`

[5] David MacKenzie, "Autoconf: Creating Automatic Configuration Scripts," `autoconf.info`.

IPv6導入編

OGATA Hiroshi ¡ogata@mma.club.uec.ac.jp¿

(1999/03/23)

1 はじめに

新入生のみなさん、電通大へようこそ。卒業までの4年間を有意義に過ごす為にも、興味のあることをサークル活動を通じて極めてみてはいかがでしょうか？

2 本題

MMAはその名の通り、メインはZ80とかi8086とかMC68000とかを対象に活動をしていた時期があったのですが、今では、すっかり、UNIX系の活動がメインになっています。FreeBSDやLinux,NetBSD等のPC UNIXや、BSD/OS,Solaris,AIX,BeOS,NEWS-OS等の商用系など、種類は様々ですが、ネットワークに繋がってなきゃ面白味も半減と言うことで、ネットワークの基本であるTCP/IPの次世代標準であるIPv6の導入手順を書いてみたいと思います。

3 環境

OSは、個人的好みでFreeBSD 2.2.8をベースにしてみました。IPv6の実装はKAMEを選んでみました。OSのinstallについては省略します。kernelを最適化したり、機能追加をしたりすることもあるので、src/sysを選択してkernelの再構築が出来るようにしておいてください。PAOが導入されていても問題ないので、Note PC等でもOKです。

4 手順

ファイルの取得

```
% wget ftp://ftp.kame.net/pub/kame/stable/kame-19990131-fbsd228-stable.tgz
```

OSに対応した最新版を使います。
既存の/usr/src/sysと、/usr/includeが書き換えられるので、心配な人はbuckupしておきます。

```
# tar zcf ip4_usr_src_sys.tgz /usr/src/sys
# tar zcf ip4_usr_include.tgz /usr/include
```

どこに展開してもかまわないのですが、/usr/src/KAMEに展開することにします。

```
# mkdir -p /usr/src/KAME
# cd /usr/src/KAME
# tar zxvf kame-19990308-fbsd228-snap.tgz
# cat /usr/src/KAME/kit/INSTALL
```

よく読んでおきます。

kernel の作成

```
# cd /usr/src/sys
# patch -p1 -f < /usr/src/KAME/kit/sys-228.diff
# cd /usr/src/sys/i386/conf
# cp myconfig myconfig.v6
# vi myconfig.v6
  options "INET6"
  options "GATEWAY6"        <= ルータとして使うときに必要。
  options IPSEC
  #options IPSEC_DEBUG
  pseudo-device bpfilter  4
  pseudo-device gif       4
```

を追加します。

```
# config myconfig.v6
# cd ../../compile/myconfig.v6
# make depend
# time nice -20 make
text    data    bss     dec     hex
1134592 73728   134584  1342904 147db8
1501.4u 115.6s 33:02.53 81.5% 929+1338k 3537+773io 390pf+0w
```

(Libretto 30 で３３分ぐらい。)

```
# make install
```

v6 アプリケーションのインストール。

```
# patch -p1 -f < /usr/src/KAME/kit/include-228.diff
# mv net net-dist
# ln -s ../src/sys/net
# mv netinet netinet-dist
# ln -s ../src/sys/netinet
# mv netkey netkey-dist
# ln -s ../src/sys/netkey
# mv sys sys-dist
# ln -s ../src/sys/sys
# mv machine machine-dist
# ln -s ../src/sys/i386/include machine
```

```
# ln -s ../src/sys/netinet6
# cd /usr/src/KAME/kit/
# time nice -20 make
1355.6u 182.5s 31:45.77 80.7% 854+889k 2787+3559io 403pf+6w
```

(Libretto 30 で 3 1 分ぐらい。)

```
# make install
```

/etc/rc の書き換え

```
# vi /etc/rc
[ -f /usr/local/v6/etc/rc.net6 ] && sh /usr/local/v6/etc/rc.net6
```

をつけ加えます。

```
# cp /usr/local/v6/etc/rc.net6.sample /usr/local/v6/etc/rc.net6
# vi /usr/local/v6/etc/rc.net6
```

設定を確認します。

man path の追加

```
# vi /etc/manpath.config
 /usr/local/v6/man
```

を追加します。

path の追加

```
% vi .cshrc
 /usr/local/v6/bin
 /usr/local/v6/sbin
```

を追加します。(/bin,/sbin より前に書かないと同名の v6 対応コマンドが使えない)

reboot します。

```
# shutdown -r now
```

起動しないときは、kernel.old, kernel.GENERIC の順に試してみます。

5 動作確認

dmesg の表示

```
ogata@libretto[31]% dmesg
Copyright (c) 1992-1998 FreeBSD Inc.
Copyright (c) 1982, 1986, 1989, 1991, 1993
        The Regents of the University of California.  All rights reserved.

FreeBSD 2.2.8-RELEASE #0: Tue Mar  9 14:56:14 JST 1999
    ogata@librettos.apple.qix.org:/usr/src/sys/compile/990309.v6
```

```
CPU: AMD Am5x86 Write-Through (486-class CPU)
  Origin = "AuthenticAMD"  Id = 0x4e4  Stepping=4
  Features=0x1<FPU>
real memory  = 21168128 (20672K bytes)
avail memory = 18399232 (17968K bytes)
PAO version: PAO-19981225
Initializing PC-card drivers: aic fdc ed ep fe sn sio wdc
Probing for devices on the ISA bus:
sc0 at 0x60-0x6f irq 1 on motherboard
sc0: VGA color <16 virtual consoles, flags=0x0>
psm0 at 0x60-0x64 irq 12 on motherboard
psm0: model Generic PS/2 mouse, device ID 0
pcm0 not found at 0x530
wdc0 at 0x1f0-0x1f7 irq 14 on isa
wdc0: unit 0 (wd0): <TOSHIBA MK0501MAT>
wd0: 477MB (977760 sectors), 970 cyls, 16 heads, 63 S/T, 512 B/S
ep0 not found at 0x300
pcic0 at 0x3e0-0x3e1 irq 11 on isa
PC-Card ctlr(0) Intel 82365A/B (5 mem & 2 I/O windows)
pcic0: slot 0 controller I/O address 0x3e0
pcic0: slot 1 controller I/O address 0x3e0
npx0 flags 0x1 on motherboard
npx0: INT 16 interface
apm0 on isa
apm: found APM BIOS version 1.1
Card inserted, slot 0
ep0: utp/bnc[*UTP*] address 00:60:97:91:a6:6a
performing DAD for fe80:0004::0260:97ff:fe91:a66a(ep0)
performing DAD for fe80:0004::0260:97ff:fe91:a66a(ep0)
DAD success for fe80:0004::0260:97ff:fe91:a66a(ep0)
nd6_dad_timer: called with non-tentative address fe80:0004::0260:97ff:fe91:a66a(ep0)
```

ifconfig の表示

```
ogata@libretto[35]% ifconfig -a
tun0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
        inet6 fe80:1::260:97ff:fe91:a66a prefixlen 64
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
        inet6 fe80:2::260:97ff:fe91:a66a prefixlen 64
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet6 fe80:3::1 prefixlen 64
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
ep0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
```

```
        inet 172.21.139.42 netmask 0xffffff00 broadcast 172.21.139.255
        inet6 fe80:4::260:97ff:fe91:a66a prefixlen 64
        ether 00:60:97:91:a6:6a
```

ping6 してみます。

```
ogata@libretto[36]% ping6 -c 3 ::1
PING6(56=40+8+8 bytes) ::1 --> ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=1.249 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.9 ms
16 bytes from ::1, icmp_seq=2 hlim=64 time=0.906 ms

--- ::1 ping6 statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.9/1.018/1.249 ms
ogata@libretto[37]%
```

とこんな感じで IPv6 による通信を行えることが確認できました。

6 今後の展開

アプリケーションのインストール/usr/src/KAME/kit/ports 以下に ports 形式で用意されているので、

```
# make install
```

で各 IPv6 アプリケーションを install します。

IPv6 マシン間の、IPv6 での通信を行う。6bone に接続して Internet を楽しむ。

7 参考文献、関連情報

KAME Project
http://www.kame.net/
IIJlab newsletter #1: 1998/05/28 「最近の IPv6」
http://playground.iijlab.net/newsletter/19980528/
IIJlab newsletter #2: 1998/08/06 「最近の IPv6 --- その 2」
http://playground.iijlab.net/newsletter/19980806/
IIJlab newsletter #5 IPv6 ほんとの話 (1)
http://playground.iijlab.net/newsletter/19981112/
IIJlab newsletter #8 IPv6 ほんとの話 (2)
http://playground.iijlab.net/newsletter/19990122/
WIDE Project IPv6 Document
http://www.v6.sfc.wide.ad.jp/v6doc/html/index.html
WIDE Project IPv6 FAQ
http://faq.v6.wide.ad.jp/index-j.html
サルでも分かったつもりの IPv6 坂根昌一

http://www.wide.ydc.co.jp/~sakane/doc/public/howto-v6.html

CSI IPv6 HOME PAGE

http://www.v6.csi.ad.jp/IPv6/

INRIA IPv6

http://www.inria.fr/Equipes/IPV6-eng.html

NTT IPv6

http://www.nttv6.net/index.html

IMASY IPv6 Project

http://light.imasy.or.jp/

ドットファイルの書き方

君島 秀征 <sanjigen@mma.gr.jp>

平成11 年 4 月 6 日

概 要

UNIX を触り始めて最初に凝るのが環境設定でしょう。ドットファイルの編集というやつです。

その全てについてはとても書けませんが、これから UNIX を使い始める人の役に少しはたてると思います。

1 一度読めば済むこと

1.1 ドットファイルとは

UNIX はマルチユーザーを想定しているので、UNIX のソフトウェアの設定ファイルは各個人ごとに独立して持てるようになっています。個人ごとの設定はユーザーのホームディレクトリに、.(ドット) で始まる名前のファイルで置かれています。これがドットファイルです。

ちなみに、daemon やネットワークの設定などマシン全体に関わる設定は別な場所 (/etc, /usr/local/etc, /usr/local/lib など) に置かれていて、一般ユーザーには変更できないようになっています。

あなた個人のドットファイルを書き換える分には他人にはまず迷惑はかかりませんし、変な設定をして起動しなくなったとしても復旧の方法はあります。怖がらずに色々いじってみてください。

1.2 ドットファイルの表示のしかた

ドットファイルは自分のホームディレクトリの一番上に置かれています。ls コマンドは、普段は. で始まる名前のファイルを表示してくれませんが、ls -A とするとドットファイルも表示されます。自分や他人のホームディレクトリを ls -A で見てみてください。. で始まる名前のファイルが色々あるでしょう?

これらのファイルはテキストファイルなので、cat や less で表示したり好きなエディタで編集したりできます。

1.3 .xinitrc と.xsession

xdm からログインしているなら.xsession だけで間に合うのですが、.xinitrc も一応用意しておきましょう。コンソールから xinit で X を起動した場合、.xinitrc が使われます。どちらも同じことをするファイルなので、cp .xsession .xinitrc しておけば十分です。でも、ln .xsession .xinitrc の方が良いでしょう。コピーではなくリンクにしておくと、片方を変更した時にもう片方の中身も変更されます。[1]

[1] 自分ちのマシンの背景はピカチュウだったはずなのに、他人に使わせたらナコルルの背景が出てきた (!) というような事故を予防します。

2 何度か読むかもしれないこと

2.1 ドットファイルの種類

大概の人のホームディレクトリにあると思われる、代表的なドットファイルを挙げておきます。

.Xauthority	xdm が作る認証情報 (設定ファイルではない)
.xinitrc	xinit で X Window System を起動した時に実行される sh スクリプト
.xsession	xdm からログインした時に実行される sh スクリプト
.xsession-errors	sh が.xsession を実行した時のエラーメッセージ (設定ファイルではない)
.Xresources	X アプリの振る舞いの設定
.Xmodmap	キー配列の設定
.twmrc	twm というウィンドウマネージャの設定
.fvwmrc	fvwm というウィンドウマネージャの設定
.cshrc	csh の設定
.login	csh が起動する時に実行される csh スクリプト
.logout	csh が終了する時に実行される csh スクリプト
.history	csh のコマンド履歴 (設定ファイルではない)
.emacs	Emacs エディタの設定
.exrc	vi エディタの設定
.rhosts	リモートログインの設定
.profile	sh/bash の設定
.bashrc	bash の設定
.bash_profile	bash の設定
.bash_login	bash の設定

以上のうち、.Xresources と.Xmodmap は、人によっては違う名前になっているかも知れません。

2.2 xdm からログインできない

2.2.1 .xsession を新しく作った後、xdm から再度ログインしようとしたら、ログインできなくなった!

これは、誰でも一度は必ず経験しています。xdm でパスワードを入力した後、[enter] の代わりに [F1] を押してください。とりあえずログインできましたね?

.xsession に実行属性が付いていないのが原因です。chmod +x .xsession としてください。次からはちゃんとログインできるはずです。

2.2.2 .xsession で xterm を起動するようにしておくのを忘れたので何もできない!

ウィンドウマネージャも xclock も起動させたのに、xterm を起動させるのを忘れたので何もできない。これも、何人かは経験しているでしょう。この場合も xdm から [F1] でログインしてください。[F1] でログインしたときは.xsession を読まずに xterm だけを起動するので、あとはエディタで.xsession を書き換えてからログインしなおしてください。

2.3 シェルを変えるには

chsh というコマンドを使います。これで、次にログインした時から、そのシェルを使えます。ちなみに、ここで指定できるシェルは/etc/shells に書かれているもののみです。

3 Xの設定

3.1 ウィンドウマネージャを変えるには

.xinitrc/.xsession を見てください。この sh スクリプトからウィンドウマネージャが起動しているので、これを書き換えます。

ただ、ここでマイナーなウィンドウマネージャを指定しておくと、あるマシンにはウィンドウマネージャがインストールされていなくて起動できない、ということもありえます。twm は X Window System がインストールされている全てのマシンに、fvwm や fvwm2 も大抵のマシンにインストールされています。

3.2 設定を変えたら一旦ログアウトして再度ログイン

.xinitrc/.xsession を書き換えた場合は、一旦ログアウトして再度ログインしないと、変更が反映されませんね。

3.3 Xアプリの色を設定しているファイルは

.Xresources というファイルです。このファイルは、X アプリの色に限らず様々な振る舞いの設定をしています。

このファイルを変更しただけでは X アプリの挙動は変わらず、xrdb ~/.Xresources などとしないと変更が反映されません。で、.xinitrc/.xsession の中で xrdb が実行されているはずです。

3.4 .xinitrc/.xsession の書き方

行末に&が付いていたり付いていなかったり、最後のコマンドだけ行頭に exec と書かれていたりします。

```
# .xsession の例
xrdb ~/.Xresources
xterm -ls &
kterm -ls &
exec twm
```

まず、他の X アプリが起動する前に xrdb が起動しています。なぜかは解りますね。次に xterm を起動していますが、行末に&が付いています。もし行末に&が付いていなかったら、xterm を終了するまで画面には kterm が現れず twm も起動しません。一度&を付けずに試してみると、意味が解ります。

twm の前に exec と書かれていますが、twm を終了したら.xsession を終了しログアウトする、という意味です。

4 csh 編

4.1 設定を変えたら一旦ログアウトして再度ログイン?

.cshrc を変更した場合は、source ~/.cshrc としたり、あるいは csh の中から再度 csh を起動したりすれば変更を反映した csh の様子が解るでしょう。

4.2 .login が読まれるタイミング

csh は、起動するときに.cshrc や.login を実行するんだけど、この二つのうち、先に実行されるのは.cshrc です。

それから、.login は必ず実行されるわけではありません。xterm に-ls オプションを付けた場合や csh に-l オプションを付けた場合くらいしか、実行されないんですね。なので、csh の設定は.cshrc に書いておいた方が良いです。

4.3 csh の環境変数

filec	ファイル名の補完をする
history	コマンドの履歴をいくつ覚えておくか
savehist	csh を終了するときに、履歴をいくつ保存するか
ignoreeof	^D で csh を終了するようにする
autologout	しばらく何も操作しないと、自動的にログアウトする
noclobber	リダイレクトで既存ファイルに上書きしないようにする
notify	バックグラウンド・ジョブの終了を即座に伝える
prompt	プロンプトの文字列

4.4 path の記述

以下のように、タブがスペースで区切って括弧で閉じます。/usr/local/{bin,sbin}のように中括弧とカンマを使うと、/usr/local/bin /usr/local/sbin に展開されます。=の前後の空白は無くても構いませんが、無くすなら=の前後の空白を両方とも無くしてください。どちらか片方だけに空白を入れるとエラーになります。

```
set path = (/usr/local/{bin,sbin} /usr/X11R6/bin /usr/{bin,sbin} /{bin,sbin})
```

行末を\でエスケープすれば複数行に分けて書けます。

```
set path = (/usr/local/{bin,sbin} \
            /usr/X11R6/bin \
            /usr/{bin,sbin} \
            /{bin,sbin})
```

同じ名前のコマンドが複数のディレクトリに存在する場合は、先に書かれたディレクトリのコマンドが優先されます。

4.5 OS ごとに path が違う

こんな風に書きます。

```
switch (`uname`)
  case "SunOS":
    set path = (/usr/local/{bin,sbin} \
                /usr/openwin/bin \
                /usr/openwin/bin/xview \
                /usr/{ucb,bin,etc})
    alias ping "ping -s \!*"
    alias ls "ls -Fg \!*"
    breaksw
  case "BSD/OS":
  case "FreeBSD":
  case "NetBSD":
    set path = (/usr/local/{bin,sbin} \
                /usr/X11R6/bin \
                /usr/{bin,sbin} \
                /{bin,sbin})
    alias su "su -m \!*"
    breaksw
  default:
    set path = (/usr/{bin,sbin} /{bin,sbin})
endsw
```

4.6 alias の設定

alias とは "別名" という意味で、長いコマンドに短い別名をつけたり、コマンドに付けるオプションの入力の手間を省いたりします。例えば、私は ls コマンドは-F オプションを付けることがほとんどなので、`alias ls ls -F` と設定しています。こうしておくと、`ls` としただけで`ls -F` が実行されるのです。定番の alias としては

```
alias ls ls -F
alias ll ls -l
alias la ls -A
alias h history
alias j jobs -l
alias su su -m
```

というようなのがあります。alias された中に alias されたコマンドがあると、その alias も展開するので、上のように設定しておくと`ll` は`ls -F -l` に、`la` は`ls -F -A` に展開されます。

私が設定している alias(の一部) も載せておきます。

```
alias reset "echo '^O^[[m' ; \reset"
alias resize "resize \!* > /dev/null"
```

```
alias beep "echo -n '^G'"
alias mcdir "mkdir \!* ; cd \!*"
alias less "less -i \!*"
alias lynx "lynx -nopause \!*"
```

4.7 complete

tcsh のファイル名の補完機能をさらに便利にするコマンドです。定番なのが`complete cd 'p/1/d/'` で、これは「cd コマンドの第1引数はディレクトリの名前しかとらないよ」と tcsh に教えているのです。こうしておくと、例えば`tmp` というディレクトリと`temp.txt` というファイルがあるディレクトリで`cd t`[tab] とした時に一発で`tmp/`と補完されます。

私は complete は詳しくないので、これ以上は触れることができません。たぶん、ikidou くんが詳しく書いてくれるでしょう。

4.8 スクリプトを分割したい

.cshrc が長くなってくると、ファイルを分割したくなりますよね。そういう時は、分割したファイルを source コマンドで.cshrc の中から呼び出してあげればいいです。alias の設定だけを.aliases などといったファイルに分割することが多いです。

4.9 csh と tcsh で条件分岐したい

tcsh には csh に無い機能が色々あるので (先に書いた complete もそう)、csh と tcsh で条件分岐させたくなることがあるはずです。簡単なのは、.cshrc と.tcshrc を用意しておくことですが、これだと似たファイルを二つ用意することになって保守性がイマイチですね。

```
if ($?tcsh) then
  complete cd 'p/1/d/'
endif
```

のように.cshrc に書いておくのが良いでしょう。

4.10 マシンによって設定を変えたい

.cshrc に

```
if (-f ".cshrc.`hostname -s`") then
  source ".cshrc.`hostname -s`"
endif
```

と書いておいて、必要に応じて.cshrc.mola とか.cshrc.kou といったファイルを作れば良いでしょう。

4.11 root の時には違うプロンプトにしたい

su -m を使っていると、root になった時もプロンプトが同じで、ちょっと不安ですよね。

```
if ( `id -u` == "0" ) then
  set prompt = "%B`hostname|sed -e 's/\..*//'` %~ # %b"
else
  set prompt = "%B`hostname|sed -e 's/\..*//'` %~ %% %b"
endif
```

という風に書いておくといいです。

最後に

ひょっとすると、これから先の人生の 1/4 を UNIX の上で過ごすかも知れないので、環境設定はちゃんとやった方が良いです。

ボヤき

ドットファイルの書き方という題だと、話題が浅くなってしまうな。それにしても、書いている内容が薄いなあ。

編集の方へ。締め切りを大幅に過ぎてごめんなさい。

しぇるのお話
～システムとユーザとの愛の架け橋～

佐藤 喬 (ikidou)
ikidou@mma.club.uec.ac.jp

1999 年 3 月末

概 要

コマンドインタプリタとしてのシェルは login から logout までの間、ユーザに最も接するアプリケーションの一つです。しかし、普段何気なく使用しているため、一生懸命働いているシェルの有難みを意識することは少ないと思います。そこで、一生懸命働かないシェルを作成し使用することにより、影の立役者であるシェルの働きぶりを見てみましょう。

1 シェルって?

Q シェルってなんですか

shell(シェル) とは簡単に言ってしまえば、「アプリケーションを実行させる機能を持つインタプリタ」ということです。つまりユーザが「何何したいなぁ」という要求をだすと、その内容を解釈し、OS(オペレーティング・システム) へ実行するようにお願いしてくれる機能をもったプログラム、ということです。

Q シェルはどうやってコマンドを実行させるの

システム・コールというものを使用して OS に要求をだします。

シェルはまずコマンドを解釈し、fork(2) をして自分の分身をつくりだし、分身に exec させます。なぜわざわざ分身をつくらねばならないかは、man execve(2) を見てみるのが良いかもしれません。

2 Stupid Shell を作る

Q Stupid Shellって何ですか

Stupid Shell(おばかなシェル) とはこの原稿用に ikidou が夜も寝ずに昼寝して[1] 製作したシェル (モドキ) です。はっきり言って使いにくいです。ソースは最後の付録につけときます。

以後ソースを読んだものとして[2] 話を進めます。

Q 使いにくいものを作って意味があるのですか

[1] 出典:つりキチ三平。
[2] 別に読まなくてもいいですが。

あります。例えば普通は自転車でいくところを歩いていったり、普通自動車でいくところを自転車でいったり[3] すれば、それぞれ自転車の、そして自動車の有難みを骨身にしみて感じるでしょう。それと同じようにこの使いにくいシェルを使えば、一般的なシェルの素晴らしさが実感できます。

3 Stupid Shell を使う

Q ls は使えるのですが cd が使えません、バグですか

いいえバグではありません仕様です。なぜ cd は使えないのでしょう。ちょっと ls と cd コマンドの実行ファイルを探して[4] みましょう。ls は/bin/ls に見つかったと思いますが、cd はファイルが見つかりませんね。これは ls は外部コマンドであるのに対して、cd はシェルの内部コマンドで実装されているからです。さて問題です。なぜ cd は外部コマンドでなく内部コマンドなのでしょうか。

ヒント、プロセスのカレント・ディレクトリを変更するシステム・コールは、chdir(2) です。もし cd が外部コマンドであり、内部で chdir(2) を呼び出しているとしたらどうでしょう。外部コマンドの cd は fork された分身でおこなわれるわけですから...

Q 環境変数を変更したいのですが、どうするのですか

できません。ついでに言えば、シェル変数においては存在すらしません。もっと言えば、内部コマンドは一つも実装していません。

Q 環境変数が使えないのにコマンド検索はおこなっているのはなぜですか

環境変数は使えないわけではありません。こちらから変更できないということです。環境変数はこの Stupid Shell を実行したシェルで設定していたものを、使用するようにしています。もしもっと不自由になってみたかったら、ソース中のある部分をコメント・アウト、コメント・インしてみてください。

Q ワイルド・カード文字が使えません

はい、使えません。UNIX においてワイルド・カード文字の展開はシェルがおこないます[5] 。このようにすればプログラマはプログラムを書くとき、ワイルド・カードの処理に時間を費すことがなくなります。いいことですね。

Q やっぱり入出力のリダイレクションも...

はい、できません。これもシェルがおこなっているのです。リダイレクションの処理をシェルがおこなうことで、stdin、stdout、stderr しか知らないプログラムでも柔軟な動作ができるのです。

Q ジョブ制御なんかも...

はい、無理です。すべて (というか一つ) のプロセスはフォア・グランドでしかはしりません。気長にまってください。

[3] 人間ってすごいですよね。
[4] which(1) コマンドを使ってみましょう。
[5] DOS の command.com とは違います

Q 他にダメなことを先に教えてください

非常に難しい質問です。列挙していけば、ソースのコメント。初期設定ができない。文字列のデリミタは空白のみ。他にもありますが、ダメダメなのがsignal処理でしょうか。例えばsleep 100(100秒スリープ)を実行し、そのプロセスをCtrl-Cで殺してみてください。どうなります?

4 最後に

どうでしたか。このおばかなシェルを使ったあとなら、普段使っているシェルの素晴らしさが実感できたと思います。現在のシェルは高機能化しており、今まで述べてきた以上に素晴らしいことができます。興味があれば自分の使用しているシェルのmanページを見てみましょう。きっと何か新しい発見があることでしょう。

5 付録

```
 1 /*
 2  * Stupid Shell (by ikidou)
 3  */
 4
 5 #include <stdio.h>
 6 #include <ctype.h>
 7 #include <unistd.h>
 8 #include <sys/types.h>
 9 #include <sys/wait.h>
10
11 #define LINE_LEN 1024
12 #define ARGC_MAX 256
13
14 char com_line[LINE_LEN];
15 char *com_argv[ARGC_MAX];
16 char *com_name;
17
18 /*
19  * get command from stdin and set com_{line,argv,name}
20  */
21 char *get_command(void)
22 {
23     char *ret;
24     int argc, col;
25
26     if ((ret = fgets(com_line, LINE_LEN - 1, stdin)) != NULL) {
27         for (argc = 0, col = 0; argc < ARGC_MAX; argc++) {
28
29             /* skip space to next an argument */
30             for (; isspace(com_line[col]); col++)
31                 /* empty expression */;
32
33             /* set this argument */
34             com_argv[argc] = &com_line[col];
35
36             /* skip this argument */
37             for (; !isspace(com_line[col]) && com_line[col] != '\0'; col++)
38                 /* empty expression */;
39
40             /* end of command line ?*/
41             if (*com_argv[argc] == '\0')
42                 break;
```

```
43              else if (com_line[col] == '\0') {
44                  argc++;
45                  break;
46              }
47              com_line[col++] = '\0';
48          }
49          com_argv[argc] = NULL;
50          com_name = com_argv[0];
51      }
52      return ret;
53 }
54
55 /*
56  * main (type ^D or ^C, and exit)
57  */
58 int main(void)
59 {
60      int pid;
61
62      printf("### Welcome to Stupid Shell World! ###\n");
63      printf("Stupid!> ");
64      while (get_command() != NULL) {
65          if (com_name) {
66              if ((pid = fork()) == 0) {
67                  /* child process */
68
69                  execvp(com_name, com_argv);
70                  /* execve(com_name, com_argv, NULL); */
71                  perror(com_name);
72                  exit(1);
73              }
74              /* parent process */
75
76              if (pid == -1) {
77                  perror("fork");
78                  exit(1);
79              }
80              /* wait child process */
81              if (wait(NULL) == -1) {
82                  perror("wait");
83                  exit(1);
84              }
85          }
86          printf("Stupid!> ");
87      }
88      printf("\n");
89      printf("### See You! :-)###\n");
90      return 0;
91 }
```

Web日記におけるエーテル実在の問題

fujita@mma.club.uec.ac.jp

概 要

「自然界の現象は全て物理法則に従っている。私達の生活に欠かせない各種の機械、装置、例えば、自転車も自動車も電車も飛行機も、電話もテレビも物理法則によって機能している。さらに地球も月も物理法則にしたがって運動している」のだそうだ。

ならばネットワークを奔流する情報も物理学で説明できるのではないか、と思いついて数分で、この駄文のあらましができてしまった。統計学や集団心理学なぞを使っての説明も考えたが、専門外のことでないことないこと振りまくのはヤバいだろうってことで、現代物理学の皮をかぶってみることに。

で、思いついたからには百萬石に書くしか、ということで書いてみたのであった。

1 情報媒介物質エーテルの存在予言

MMA 内部で日記文化が隆盛を極めておよそ一年になる。この間、MMA 部員のネットワーク度はますます進行し、世間では PHS や携帯電話による口頭での情報伝達が主流なのをしりめに、Web 日記に書いた方が速く情報が伝播するという奇妙な現象が生じた。その速度の素早さから、日記情報を媒介する仮想物質エーテルの存在が提唱される。そして日記エーテルは MMA 内部のみならず各地に何だかよくわからない影響[1] を及ぼし、現在も各人の bookmark を増大させ続けている。

以下の文は、その日記エーテルの振るまいを記述しようというバカな企画を試しにやってみたものだが、はっきし言って支離滅裂な内容である。それでもたまにはこんなヘンな記事も良いのでは... とか思ったりする。

2 特殊相対性理論を使ってみる

2.1 Einstein の相対性原理と Lorentz 変換

Einstein の相対性原理とは、

1. すべての慣性系はまったく対等である。
2. 真空中を伝わる光の速さはすべての慣性系において一定である。

という二つの基本原理のことで、これらを用いて特殊相対性理論が組み立てられる。で、この原理を用いると、ある座標系 $S(O-xyz)$ と、S から見て一定速度 V で移動する $S'(O'-x'y'z')$ について、
$x' = \gamma(x - Vt), y' = y, z' = z, t' = \gamma(t - \frac{Vx}{c^2})$、(ただし、$\gamma = \frac{1}{\sqrt{1-(\frac{V}{c})^2}}$、c:光速度) という変換が成り立ち、これを Lorentz 変換という。

さらに Lorentz 変換によって、物体が伸縮する Lorentz 収縮や、浦島効果で有名な時間の遅れといった現象が示せるワケだが、どうもこの辺はおちょくり甲斐がないらしく、例えば慣性系をパケットとみなすと、パケットが光速度を越えないのはまだ理解できないでもないが、パケットが縮んだりするのは直感的に理解不能なので、この辺りはあんましイジらないことにした。

[1] 某 hauN のマジカル某とかが好例?

2.2 光速度の意味

ある Web 上の情報が原因となり、別の Web ページに結果として反映される場合、どの座標系からみても結果は原因よりも後でなければならないという因果律が成り立つ。
座標系 S の Web ページ x_1 において時刻 t_1 に掲載された情報と、x_2 において時刻 t_2 に掲載された情報を S' で観測すると、二つの事象の時間間隔の間には、$t_2' - t_1' = \gamma(t_2 - t_1 - \frac{V}{c^2}(x_2 - x_1))$ という関係が成り立ち、この式において $t_2 > t_1$ のときにつねに $t_2' > t_1'$ であるためには不等式 $1 - \frac{V}{c}\frac{x_2 - x_1}{c(t_2 - t_1)} > 0$ が成り立たねばならない。このための必要十分な条件は $|\frac{x_2 - x_1}{t_2 - t_1}| \leq c$ である。これは因果律によって結ばれる現象は光速度より速く伝わることはできないことを表している。即ち真空中の光速度は情報の伝わる速度の上限なのである[2]。

3 Schrödinger 方程式も使ってみる

3.1 Web 上の Schrödinger 方程式

Web 上の情報の波動性 (笑) を記述する関数 ψ を与える方程式は Schrödinger 方程式と呼ばれ、定常状態の Schrödinger 方程式は $-\frac{\hbar^2}{2m}(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} + \frac{\partial^2\psi}{\partial z^2}) + V(x,y,z)\psi = E\psi, V(x,y,z) = -\frac{1}{4\pi\varepsilon_0}\frac{e^2}{\sqrt{x^2+y^2+z^2}}$ と表される。これを三次元極座標に変換して解くと、波動関数 $\psi = \psi(r,\theta,\phi)$ は $\psi_{nlm_l} = R_{nl}(r)\Theta_{lm_l}(\theta)\Phi_{m_l}(\phi)$ と書くことができる。$n = 1, 2$ に対する波動関数の具体的な形は

n	l	m_l	$R(r)$	$\Theta(\theta)$	$\Phi(\phi)$
1	0	0	$\frac{2}{\sqrt{a_0^3}}e^{-r/a_0}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2\pi}}$
2	0	0	$\frac{1}{\sqrt{8a_0^3}}(2 - \frac{r}{a_0})e^{-r/2a_0}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2\pi}}$
2	1	0	$\frac{1}{\sqrt{24a_0^3}}\frac{r}{a_0}e^{-r/2a_0}$	$\sqrt{\frac{3}{2}}cos\theta$	$\frac{1}{\sqrt{2\pi}}$
2	1	± 1	$\frac{1}{\sqrt{24a_0^3}}\frac{r}{a_0}e^{-r/2a_0}$	$\frac{\sqrt{3}}{2}sin\theta$	$\frac{1}{\sqrt{2\pi}}e^{\pm i\phi}$

となり、情報が取得可能な人数 $n = 1, 2, 3, \cdots$、ルータ数 $l = 0, 1, 2, \cdots, n-1$、経路数 $m_l = -l, -l+1, \cdots, -1, 0, 1, \cdots, l-1, l$ である[3]。

3.2 Web 情報の確率分布

Schrödinger 方程式の解である波動関数の物理的な意味は、その絶対値の二乗がその人間に Web 上の情報が届く確率密度関数を与えるということである。情報が $r \sim r + dr$ の範囲に届く確率 $P(r) = r^2|R_{nl}(r)|^2$ であり、図で描くと正規分布のグラフのよーになるのだが、gnuplot とかを使いこなす自信はないので、どんなふうになるか是が非でも知りたい向きは書店で量子物理学の本でも立ち読みすると良い。

参考文献

[1] 伊東敏雄,「な〜るほど!の力学」, 学術図書出版社

[2] 平成 10 年度昼間コース第三学期情報工学科, 現代物理学の講義

[2] これはパロディではなく物理学的に本当のコトであったりする...
[3] ホントは主量子数 n, 角運動量量子数 l, 磁気量子数 m_l である。

GRobot

uirou@mma.club.uec.ac.jp

1999/03/22

概 要

決して広大とは言えないフィールドに、「それ」は存在していた。同じ能力をもった「相手」を破壊するためだけに。

Computer-A: scan...done. (243,104) にロボット。識別符号 243。敵と認識。現在の速度から敵の未来位置を予測…

Computer-A: missile(340,150,3). ミサイル射出を確認。

Computer-B: thrust. missile.

Computer-A: scan...done. 3 時方向からミサイル。接近中。回避の必要を認む。

Computer-A: thrust(4,3,2,3) 加速開始。

Computer-B: missile. missile. missile. thrust.

Computer-A: scan...done. ミサイル大量に接近。迎撃。

Computer-A: missile(120,153, 6). thrust(-2,3,-4,2). 迎撃用ミサイル射出、回避。

Computer-A: scan...done. 接近中のミサイル消失を確認。

Computer-A: missile. 反撃開始。

1 GRobot とは？

GRobot とは、grobot と呼ばれる、C 等の言語によってその動作を (あなたによって！) 記述されたロボット同士がリアルタイムに動き、策敵し、攻撃して、互いに破壊しあうというゲームです。

ユーザは grobot の動作を考え、記述する事によってゲームに参加します。実は、grobot は

登録, 移動, 策敵, 攻撃

という、たった四つの命令で動かす事が出来ます。[1] たった四つとはいえ、その用法によってさまざまな戦術が考えられるので、とっても奥が深いゲームであるとも言えます。

ということで、ここではその GRobot についてのレビューを書いてみたいと思います。

2 何が出来るの？

では、ロボットの動作を決めているそれぞれの命令についてざっと[2] 眺めてみましょう。

登録 (gr_init) あなたのロボットをゲームに参加させます。この時にロボットの名前や所属するチームの ID を登録できます。

移動 (gr_thrust) ロボットを移動させます。ロボットには上下左右に 4 つの加速器がついており、その噴射口の力と噴出時間を指定できます。

[1] 登録は最初の一回だけしかする必要が無いので、実際には移動, 策敵, 攻撃の三種類だけということになります。

[2] 詳しく説明していると紙面が無くなってしまうので

攻撃 (gr_missile)　ミサイルを撃ちます。目標となる座標とミサイルのミサイルの爆発力を指定できます。しかし、ロボットの一度に撃てるミサイルのパワーには限度があるので、加減が重要です

策敵 (gr_scan)　他のロボットや発射されたミサイルをスキャンします。わかるのは

- 座標
- 速度
- ロボットのダメージ又はミサイルの爆風の範囲
- ロボットやミサイルの ID
- ロボットやミサイルのチーム ID

です。

3 Welcome to GRobot World

さて、ロボットがどんなことが出来るかがだいたいわかったので、次は実際にどのように動くのかをみてもらいたいのですが、紙面ではそうもいきません。仕方が無いので GRobot の世界での物理法則がどのようになっているかを箇条書にしてみたいと思います。

- GRobot の世界には摩擦がありません。一度動き出した物体はいつまでも動き続けます
- GRobot の世界には完全弾性体しか存在しません。同じ質量の物体が衝突すると、その速度がそっくり入れ替わります
- ロボットは爆発に巻き込まれる事と物体に衝突する事によってのみダメージを受けます
- ロボットが戦うフィールドは正方形で、四方に壁が存在します。この壁を破壊する事は出来ません
- 弾は着弾地点に到達するか、ロボットに命中するか、爆風に巻き込まれるまで爆発しません
- 弾は加速できません。一度射出されたら一定の速度で着弾地点へ飛んで行きます
- ロボットは爆風に巻き込まれると吹き飛ばされるほど軽いです
- 弾のパワーは弾が爆発し終えると回復します
- 弾はパワーが大きくないと大きな爆発を起こしません

と、いった感じです。これらのことから、

- 弾は軌道が簡単に算出できる
- 弾を避けるには自分で弾を撃ち、爆風に巻き込んで破壊する方法もある
- 弾は乱射できない
- 壁際に位置していると爆風で壁に叩き付けられる可能性がある

ということが予測できます。それらのことから、たとえば次のような戦術が考えられるでしょう。

- とにかく接近して確実に命中させる
- 接近して来る弾をパワーの低い弾で相殺しつつ、正確に相手を射抜く
- うまいこと壁際においつめる

もちろんこの他にもいろいろな戦術が考えられますね。

4 戦術の基礎

GRobot の目的は、「最後まで生き残る事」なわけですから、「ダメージを食らわないようにする」のが第一です。それには「飛んで来る弾をスキャンし、回避する」必要があります。それも、発見が遅れると被弾する可能性が高まってしまいますから、しつこいくらいスキャンした方がが良いかもいしれません。

もちろん敵を倒さない限り「最後」にならないわけですから、敵を攻撃する必要があります。しかし、敵の未来位置を予測するのはそうそう簡単に出来る物ではありません。そこで、命中精度を高めるために「接近する」のが効果的だと思います。が、接近すると敵からこちらへの命中精度も高まるので注意が必要です。出来れば遠距離から敵に確実に弾を当てる方法を考えたいものです。

また、古来からスペースハリアーなどで用いられてる「円運動で敵の攻撃を回避する」という手法は GRobot でも有効な戦術のようです。円運動をすることによって自分の位置を常に移動させ、加速する速度や方向を常に変化させる事によって敵からの攻撃を効率的に回避できるからです。

これ以上戦術について言及してしまうとみなさんの楽しみを奪ってしまう事にもなるのでこのくらいにしておきます。[3] また、GRobot における戦術は、まだまだ確立されておらず、実際には暗中模索といった所のような気がします。例えばチーム ID による仲間の識別は、かなり難しいですが、使い方によっては強力な連係プレイが期待できるでしょう。

5 GRobot の歴史

GRobot は CRobot という C 言語を用いる同じようなゲームを元にして作られたそうです。実際に作ったのは僕ではなく uncover さんで、過去の百萬石にはその記事が載っていたと記憶しています。で、現在は僕が GRobot のメンテをやっているのですが、実は CRobot というものは話にしか聞いた事がありません。ですので、実は GRobot の最終形態というか、CRobot の内容はさっぱり知らないのです。ということなので、実は僕の趣味で勝手に拡張を加えている所も多く、多分元になった CRobot とは既に似ていない物になっているかと思います。というか、別に CRobot に似せなければいけないという事は何処にもないので、何か楽しい変更依頼があったら教えて下さい。歓迎します。

6 GRobot の目指すもの

GRobot は、そのシンプルな所が売りです。[4] ロボットに与えられる命令はたったの四つですし、それへの引数も最小限度しかありません。また、衝突などもわかりやすいように完全弾性衝突です。唯一わかりにくくなっていそうなのは爆風による加速でしょうか。これによって厳密な計算はあまり効果が無くなっているかもしれません。といっても、これですら計算できないわけではありません。grobotd は乱数など何処にも使っていないのですから。[5]

ということなので、GRobot のライブラリ、すなわちロボットへの命令セットへの変更は出来るだけ最小限に押えたいとか思っています。ですが、5 節で書いたように面白い変更は歓迎します。いつでも言って来て下さい。

---

[3] 僕が考えつかないなどと言う事が真実であるのは君と僕の秘密だ！
[4] というか、シンプルで、C 言語などの学習用に使える、というのが真の売りです。
[5] でも float は使用しているので真に厳密な計算は出来ていませんね

7 バグ

ここに書かれている事で GRobot には実はまだ実装されていない機能があったりするかもしれません。

おまけ：弾を避ける

scan して得られた弾の軌道が、自分の位置を狙っていたら、避けなければいけません。ということで弾を避けるプログラムのサンプルを示しておきます。

```
#include <stdio.h>
#include <math.h>
#include "grobot.h"

int avoid(SCAN *my, SCAN *missile){
  double x, y;
  double vx, vy;
  double tx, ty;

  if(my == NULL || missile == NULL)
    return(0);

  if(missile->obj != SC_MSL)
    return(0);

  /* 相対位置を x,y に入れる */
  x = missile->x - my->x;
  y = missile->y - my->y;

  /* 相対速度を vx,vy に入れる */
  vx = missile->vx - my->vx;
  vy = missile->vy - my->vy;

  /* 遠ざかっているようなら何もしない */
  if((x > 0 && vx > 0) ||
     (x < 0 && vx < 0) ||
     (y > 0 && vy > 0) ||
     (y < 0 && vy < 0))
    return(0);

  /* でなければ近付いているのだから、+90 度方向に加速する */

  { /* 速度成分を 90 度二次元回転する */
    double s,c,dummy,r;
    r = (double)90 / 360 * M_PI;
    s = sin(r);
```

```
    c = cos(r);
    dummy =(c*vx - s*vy);
    vy =(s*vx + c*vy);
    vx = dummy;
  }

  gr_thrust(vx, 1, vy, 1);

  return(1);
}
```

実はこのプログラムをそのまま使うと、壁に激突してしまいますし、敵に接近する行動をとらせようとするのも難しくなります。ということでこのまま使うのは問題があるでしょう。

私家版 Note PC TIPS

君島 秀征 <sanjigen@mma.gr.jp>

平成11 年 4 月 6 日

概 要

新しい Note PCって良いですよね。ところで、Note PC を買ったらまず最初にすることは何でしょう。分解? 同機種を使っている人の使用レポートを探すこと? それとも保守マニュアルの注文? 大抵の場合、バックアップと OS のインストールですよね。

ここでは、バックアップと OS のインストールについて、"こんな方法もあるよ" といったことを書いてみます。

この文書のねらい

大学入学祝いに Note PC を買ってもらっちゃったかも知れない新入生への、PC-UNIX on Note PC の誘い。あるいは、誘われた人が金/時間/労力をあまり無駄に使わないように導くこと[1] 。

私が今よりももっと若かった頃は、FreeBSD の CD-ROM の中身を、PC-9801 から AcerNote の FAT 領域にシリアルポート経由で転送して、そこからインストールしたりしたものです[2] 。今思えば、あれは時間/労力の無駄でした。この文書を読む人には、少なくとも、あれだけの時間/労力の浪費は避けさせてあげたいです。

というわけで、金を浪費した挙げ句、2 枚も PCMCIA SCSI カードを持っている君島がお送りします[3] 。

とりあえず "こんな方法もあるよ"ってことだけ知っておけばいいです。"どうやるの?" は、その時に周りにいた人に聞いてください。

バックアップ編

初めて Windows が起動したら、フロッピーディスクを 30 枚用意してバックアップを取ってくれ? そんなことやってられるか。

dd でバックアップだ

UNIX には、例えば 1GB の HDD を、そのまんま 1GB のファイルとして扱うような方法が用意されてます。フロッピーディスクを読んで 1.44MB のファイルに書き出したり、あるいは逆のことをするツールっていうのがありますよね。あれと同じことが HDD に対してできるんです。これを使えば、Windows も何もかもそのままバックアップできます。

フロッピーディスクから UNIX を起動して、HDD の中身を吸い出しつつ圧縮し、ネットワークを介して別のマシンのディスクに HDD の中身をバックアップしたりできます。

[1] 時間/労力を無駄にすることは必ずしも無意味ではないんだけど、金は使わずに済むに越したことはないでしょ。
[2] マジで一晩かかったっすよ。
[3] SCSI カードを 2 枚持っているのも、無知だった故。

HDD を取り外せ

ネットワークを使うのが面倒くさい場合、あるいはNote PC の FDD がちょっと変だったりする場合[4] は、HDD を取り外してデスクトップ機に繋いで dd でバックアップです。Note PC 用の HDD も、コネクタがちょっと違うだけで IDE です。変換アダプタを挟めばデスクトップ機に繋がります。

インストール編

LASER5[5] に CD-ROM を買いに行く? UNIX USER[6] の付録 CD-ROM に収録されるのを待つ? Plat'Home[7] に Release&Write CD-ROM を買いに行く? そんな必要ノンノン。

情報収集しましょう

重要です。サーチエンジンを上手に使えば、手間が掛からない割に得られる物は大きいので、インストールする前には必ず下調べするべきです。

それと、カタログや説明書でビデオチップの種類と VRAM の容量を調べておくと後で楽です。

Windows を使う

Windows が入っているのなら、IRQ や I/O ポートや DMA の使われ方を事前に調べておきます。

ネットワークインストールだ

PC-UNIX のインストーラには、フロッピー一枚で boot して、あとの必要なファイルはネットワークから落としながらインストールする機能が大抵あります。CD-ROM を買ってくる必要は無いですし、CD-ROM ドライブを Note PC に繋ぐために投資したりする必要はありません。それに、雑誌の CD-ROM よりネットワークにある物の方が常に新しいです。

学内にミラーサイトがあるぞ

電通大には FreeBSD の公式ミラーサイトや NetBSD の非公式ミラーサイトがあります。これらを利用すると、とても速くインストールができます。

隣のデスクトップ機をサーバーに

どうしても CD-ROM からインストールしたい場合でも、Note PC に CD-ROM ドライブを繋ぐくらいなら、デスクトップ機に CD-ROM を読ませて、そこからネットワークインストールする方がきっと楽です。

---

[4] FDD を PCMCIA 経由で接続するのでネットワークカードと同時に使えないとか、そもそも FDD が別売で俺は来月まで買えないんだ、とか。

[5] 秋葉原の CD-ROM 屋。Linux Japan という雑誌の出版元でもある。

[6] ソフトバンクの UNIX 雑誌。

[7] 秋葉原のハード/ソフトなんでも屋。最近はレゴブロックも扱っているらしい。

HDD を取り外せ

HDD を取り外してデスクトップ機に繋いでしまえば、デスクトップ機のネットワークアダプタや FDD や CD-ROM ドライブが使えます。HDD が簡単に取り外せる機種なら、これが最も楽です。

FAT から boot. FAT からインストール

FDD が変で、うまく boot しない、しかも HDD を取り外すのは非常に骨だ、というような場合もあります。その時は、Windows の力を借りて FAT パーティションに配布物を置いて、DOS 用のブートローダで boot してインストールしましょう。

機器編

物は大抵借りられる

必要な物は、誰かが必ず持っているので借りられます。喜んで貸してくれる人もいます。インストール時にしか使わないのなら、借りるだけで間に合いますよね。

Ethernet カードは買おう

前述と矛盾するかもしれませんが、Ethernet カードは買いましょう。日常的に使う物ですから。安い物なら 5,000 円くらいですし。

SCSI カードは要らない

ほんと、使わないです。

最後に

最終的には、うまくいきます。試行錯誤も楽しみのうちだと思ってください。

追伸。要らない {Think|Work|Cross}Pad 買います。

アプリケーションを IPv6 ready にしよう
～いつかくる日に慌てない為に～

楯岡孝道
(tate@spa.is.uec.ac.jp)

1 はじめに

現在用いられている IP version 4 (IPv4) のアドレス空間の枯渇が叫ばれて久しい。この問題によりプライベートアドレスや NAT などの技術が発達したが、本質的解決には IP version 6 (IPv6) への移行が必要である。

IPv6 技術は未だに細部に関して議論が続けられている技術であるが、既に複数の実装の相互接続が行なわれ、世界的な実験ネットワークが稼働している熱い技術でもある。

ネットワークが稼働しても、アプリケーションが IPv4 しか扱えないのでは意味が無い。そこで、本稿では従来の TCP/IP アプリケーションを IPv6 対応にする方法の基礎を述べる。

2 getaddrinfo(3)

従来の gethostbyname(3) は一つのホスト名に対して一種類のアドレスタイプ (AF_INET) しか応答できないという問題があった。しかし、IPv6 の世界では IPv4 と IPv6 の両方を実装したデュアルスタックホストが多数存在するため、V4/V6 を合わせた全ての利用可能な全てのアドレスに対して接続を試みるような機構が必要である。

そこで IPv6 においては、これらの発展形であり、アドレスタイプ等まで含めた addrinfo 構造体の配列を返す getaddrinfo(3) [2] が利用される。

3 実際の実装

基本的には、gethostbyname(3) と getservbyname(3) を getaddrinfo(3) で置きかえ、そこから得られた複数のアドレスにループによって順番に接続を試みるのが一般的な実装法である。

注意すべきは、socket(2) で作成するソケットはアドレスファミリによって異なるため、ループ中で、getaddrinfo(3) で得られた各 addrinfo 毎に ai_family の値を用いてソケットを作成しなおし、接続に失敗した場合には破棄するという実装にする。

すなわち、接続部分は以下のようになる。

- getaddrinfo(3) でアドレスの列を取得
- 各アドレスに対して
  - ai_family のソケットを作成
  - connect により接続する

– 成功した場合にはループを終了
– 失敗した場合にはソケットを破棄

このコードは特定のアドレスファミリに依存していない、すなわち IPv6 に限らず他のアドレスファミリが混在していても利用可能となる。

残念ながら getaddrinfo(3) は IPv6 をインストールしないと用意されないが、ssh の IPv6 化実装 [3] には getaddrinfo(3) の互換関数が含まれるので、システムに getaddrinfo(3) が存在しない場合にはこれを利用するようにするのが最もスマートな方法であろう。

4 おわりに

今回はアプリケーションの IPv6 化のごく基礎のみを書いたが、この技術だけで多くのアプリケーションは IPv6 化が可能である。このようにしたアプリケーションはもちろん IPv4 でも利用可能である。

「まだ 2000 年には年数があるから」と考えたプログラマによって 2000 年問題は生じた。自分のアプリケーションを IPv4 の制限から解き放つのに早すぎることはあるまい。

参考文献

[1] Jun-ichiro itojun Itoh, "Implementing AF-independent application,"
`http://www.kame.net/newsletter/19980604/`

[2] R. Gilligan, et al., "Basic Socket Interface Extensions for IPv6," RFC2133.

[3] KIKUCHI Takahiro, "ssh-1.2.26-IPv6 version 1.4,"
`ftp://ftp.kyoto.wide.ad.jp/IPv6/ssh/`