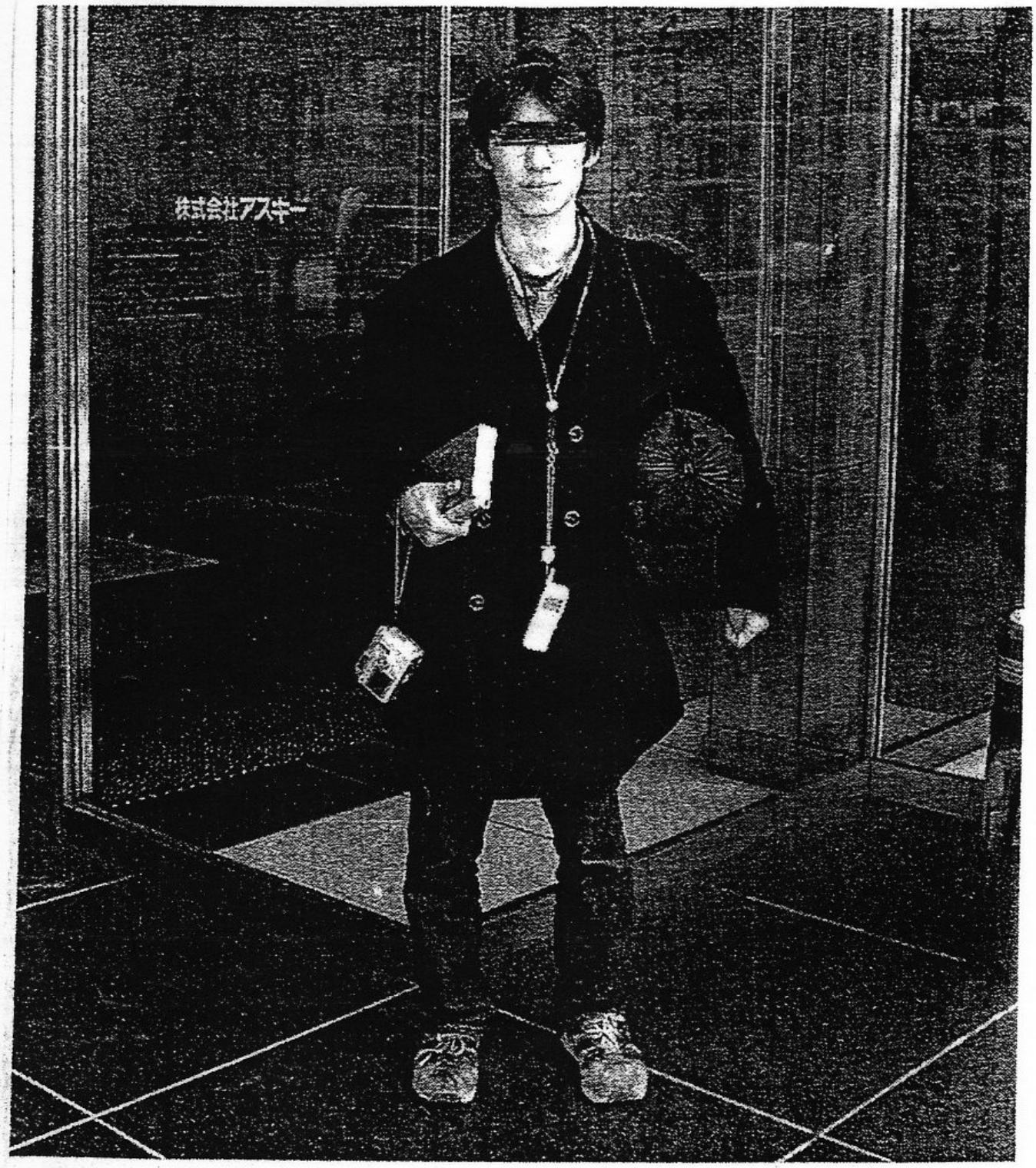

写真3　装備を身に付け、Ah!SKI社前にたたずむ原田。デーモンホーンがよく似合うぞ。さぁ、旅立ちの時は来た。

百万石 2001春

新入生歓迎の辞

sakura@mma.club.uec.ac.jp.

2001 年 4 月

新入生の皆さん入学おめでとうございます。

MMA は ”Microcomputer Making Association” の略称です。以前は MMA の名称通りにコンピューターの設計・製作をしていたようですが、最近では高性能なワークステーションやパソコン等が安く入手できるようになったので、コンピューターの製作はしておらず、AT 互換機の組み立てをするにとどまっています。

現在の主な活動内容は、UNIX 系 OS 上での、C、C++、JAVA、Perl、Ruby 等によるソフトウェア開発やネットワークの保守・改良等です。また、MMA に入れば、パソコンを部室に持ってきて、プライベート IPv4 アドレスを割り当て、インターネットに接続することができます。さらに、MMA では、IPv6 の試験運用もしているので、グローバルな IPv6 アドレスを割り当てることにより、V6 的に世界中からダイレクトにアクセスできるようになります。

入部に際して、知識や経験は有った方がいいですが、無くても全く問題ありません、人生はまだ長いのでこれから身につけていっても、人並み以上にはなれるはず。必要なのは興味とやる気 (と入部金) です。

MMA に興味を持った人は、サークル会館 2 階の部室まで見学に来てください、入部を迫ったりすることは無いので安心してください。見学は随時可能ですが、部室に人が居ないこともあります、そういう時もめげずにまた来てくださいね。なお、部室に人が居る確率が高いのは、昼休みと 4 限以降です。

一年生以外でも大歓迎です、誰でも遠慮せずに見学に来てください、お待ちしています。

DHCP

sakura@mma.club.uec.ac.jp.

2001 年 4 月

1　序文

3 月 9 日
詳しい日付は覚えてないが、確か 2 月に abe くんに DHCP の事を書かない？とか言われた、きっかけもさっぱり覚えてない。とにかく DHCP 書くことになった。
基本は RFC でしょ、とか言われたのでさっそく、ウェブ上を検索してみた、なるほど DHCP は RFC2131 を基にしてるんだな、さっそく英文をよんでみよう。なんかやたら長いな、日本語訳を探そう。JPNICE[1]だめじゃん、2131 の和訳出てないよ、まいいや、他を探そう、おおあった[2] 、これを読めば DHCP はいただきだな。でもなんか長いから後で読もう。
ひとまず DHCP を分かりやすく説明してるページ探そ、おー！これは[3]、なんだこのページは、困るなこういうページ作られちゃ、分かり易すぎるんだよね、これじゃ僕が調べることないじゃん、
4 月 3 日
やばい、書こう書こうと思っているうちに、いつの間にか新歓直前になってしまった、こんなはずじゃなかった... とか言っていても始まらない、今日書き上げるしかない。という決心をした。
4 月 4 日
そうこうしているうちに、いつの間にか過去と未来が交差していた。

2　初めに

皆さんはノートパソコン等を LAN(Ethernet) に接続したことがあるでしょうか？接続したことが無い人の為に説明しておきますが、ケーブルをハブと呼ばれる箱に挿すだけでインターネットに接続できるようになります。便利ですね、なぜこんなことができるのでしょうか？
さっそく先ほどの疑問にお答えしましょう、これを実現しているのは DHCP です。今回はこの便利な DHCP の仕組みをほんの少しだけ紹介します。

3　DHCP とは？

DHCP は、Dynamic Host Configuration Protocol　の略で、TCP/IP ネットワークに接続されているコンピュータに、IP アドレスや、ネームサーバーのアドレス等の様々な設定情報を動的に設定するための決まりです。

4　何ができるの？

まず先ほど書いた事、即ち、ケーブルを挿すだけでインターネットに接続できます。しかしこれでは、あまりにも大雑把すぎますね。そこでもう少し詳しく説明すると、インターネットに接続するための設定を自動的にすることができる、ということができます。
実は DHCP はとても高機能で、インターネットに接続するための設定だけでなく、他にも多くの設定をすることが可能です、多いというだけでは良く分からないと思うので、具体例をいくつか挙げておきます。

1　主な設定項目

`tmof`：世界標準時刻との差 (秒)(日本は 32400)
`snmk`：割り当てる IP アドレスのサブネットマスク
`brda`：ブロードキャストアドレス
`rout`：ルーターのアドレス
`dnsv`：ネームサーバーアドレス
`dnsd`：ドメイン名
`ipad`：クライアントに割り当てる IP アドレス

[1]仮名です
[2]http://www.bekkoame.ne.jp/ poetlabo/LIBRARY/rfc2131j.txt
[3]http://www02.so-net.ne.jp/ hourin/jus980422/

`dfll` : クライアントからの要求が無い場合の有効期限（秒）
`maxl` : 割り当てアドレスの最長有効期限 (秒)
`clid` : クライアントの識別子 (`WIDE` 版では `MAC` アドレス) を指定する
`hstn` : クライアントホスト名
`nisd` : `NIS` ドメイン名
`nisv` : `NIS` サーバの `IP` アドレス

他にもまだまだたくさんありますが、それらを挙げると説明が進まないので、最後に書いておきます。

5 自動設定の仕組み

何ができるか分かってきたところで、次はどのように設定が行われるのかを説明します。
DHCP はクライアント・サーバ方式で、DHCP サーバーがネットワークアドレスを動的に割り当て、設定情報とともにホストに送信します。ここでこのクライアント・サーバ間では、DHCP メッセージをやり取りすることによって通信を行っています、また、サーバは複数の場合もあります。
それでは、クライアントがネットワークに接続してから IP アドレスを割り当てられ、最後に割り当てられた IP アドレスを返還するまでの流れを見てみましょう。

1. クライアントをネットワークに接続する。
2. クライアントはサーバを見つけるためにブロードキャストする。(DHCPDISCOVER)
3. サーバは DHCPDISCOVER への応答として、クライアントへ設定情報を送信する。(DHCPOFFER)
4. クライアントは DHCPOFFER を送信したサーバからを一つ選択し、受信した設定情報の割り当てを要求するために、サーバ ID を含めてブロードキャストする。これにより、選択されなかったサーバは別のサーバが選択されたことを知ることが出来る。(DHCPREQUEST)
5. 選択されたサーバはクライアントに要求された設定情報を送信する。(DHCPACK) 選択されなかったサーバは DHCPOFFER でクライアントに送信したネットワークを利用可能に戻す.
6. クライアントは DHCPACK を受信し設定を完了する。
7. クライアントはサーバに DHCPRELEASE を送信することによって、割り当てられたアドレスを放棄する。

以上が自動設定の大まかな流れです。

6 実装について

さて、DHCP が何なのか分かってきて、実際に DHCP を使ってみたくなったと思うので、UNIX 系 OS で使われているフリーのものを 3 つ紹介します。

1. WIDE 版 DHCP[4]: これを使っている人の多くは BSD 系の OS を使っています。
2. ISC 版 DHCP[5]: Linux 系の OS を使っている人の多くがこれを使っていると思います。
3. CMU 版 DHCP[6]: 詳しいことは分かりません。

今回は個々のパッケージの使用法についての解説はしません。
あとは、自分でインストールや設定をして楽しんでください。

[4] http://www.wide.ad.jp/software/index-j.html
[5] http://www.isc.org/products/DHCP/
[6] http://wwwnet.princeton.edu/software/dhcpd/

7　おわりに

僕はこれを書くまで DHCP のことをほとんど知りませんでした、実は今でもあまり良く分かっていません、が少しでも DHCP の雰囲気を味わって頂けたとしたら幸いです。
それでは、よい学生生活を。

参考文献

RFC: DHCP Options and BOOTP Vendor Extensions
1533: (Obsoletes: 1497, 1395, 1084, 1048)
2132: (Obsoletes: 1533)

DHCP Options for Novell Directory Services
2241:

Dynamic Host Configuration Protocol
1531:
1541: (Obsoletes 1531)
2131: (Obseletes 1541)

URL: DHCP の仕組み、設定方法等
http://www02.so-net.ne.jp/~hourin/jus980422/
http://www.sfc.wide.ad.jp/~tomy/dhcp/
http://www.db.is.kyushu-u.ac.jp/~kimura/freebsd/freebsd.html#settei-network

DHCP サーバー配布関連
http://www.wide.ad.jp/software/index-j.html
http://www.isc.org/products/DHCP/
http://wwwnet.princeton.edu/software/dhcpd/

和訳 RFC
http://munch.c.u-tokyo.ac.jp/net/rfcjapan.html
http://www.net-web.co.jp/ipusiron/source/RFCJ.htm

付録

RFC で決められている設定項目。
但しコードが - となっているものは RFC で規定されていません。

WIDE 版 DHCP のマニュアル（dhcpdb.pool.5.jis）から引用

・ここで使用するタイプには次のような意味がある。

bool	"true"または"false"のどちらかの値をとる。これらと ことなる文字列を与えた時には、false とみなす。 (ex. ipfd=true)
str	文字列。コロンなどを利用したい場合には、ダブル クォート (") などで括る。 (ex. hstn="clapton")
octet	整数 (8bit) (cf. 8, -1, 070, 0xff などの 10,8,16 進数が扱える)
short	整数 (16bit)
long	整数 (32bit)
ip	単一の IP アドレス。(a.b.c.d, a.b.c, a.b のどれかの形式)

(ex. ipad=133.4.31.9)

iplist　複数の IP アドレスのリスト。各々の IP アドレスは空白で区切られる。
(ex. rout=133.4.31.1 133.4.31.2 133.4.31.3)

ippairs　IP アドレスの組のリスト。各々の IP アドレスは空白で区切られ、2 つずつ組にされる。
(ex. strt=133.4.27.0 133.4.31.1 133.4.36.0 133.4.31.1)

mtpt　整数 (16bit) のリスト。各々の整数は空白で区切られる。
(ex. mtpt=1 2 4 6 8)

clientid　クライアントの識別子。"type:data"というフォーマットを持つ。type は 0〜255 までの整数であり IANA によって定義されている。data は本来は自由なバイト列なため、ここでは各バイトを 2 桁の 16 進数であらわす。通常は MAC アドレスを記述することが多い。
(ex. clid="1:0x08004600e5d5")

・コードは、RFC2132 に規定されているオプションのコード番号である。したがって、RFC2132 を参照して欲しい。またコード番号が'-' となっているものは、そのフィールドが RFC2132 に規定されたオプションではないことを示す。

名称	コード	タイプ	内容
tblc	-	str	別エントリへの参照。参照されたエントリが引用される。
snam	-	str	DHCP メッセージの sname フィールドを定義。DHCP サーバのホスト名を記述可能。
file	-	str	DHCP メッセージの file フィールドを定義。boot イメージのファイル名を記述する。
siad	-	ip	boot イメージを供給する tftp サーバのアドレス。
albp	-	bool	true の場合、このエントリは BOOTP クライアントに対して割り当てても良いことを示す。静的なエントリの場合には、デフォルトで true となる。true の場合、maxl も無限大に設定される。
ipad	-	ip	割り当てる IP アドレス。
maxl	-	long	許容する最大の割当期間 (lease) を秒数で記述。本実装のデフォルトは 3600 秒 (database.h で定義)。
dfll	-	long	デフォルトの割当期間。クライアントからの割当期間の希望値がない場合は、この値が使用される。本実装のデフォルトは 3600 秒 (database.h で定義)。
clid	-	clientid	静的な割当の場合に、クライアントの識別子 (Client Identifier) を記述する。本実装に含まれる DHCP クライアントでは MAC アドレスを用いるので、割当対象が WIDE 版クライアントの場合にはここに MAC アドレスを記述する。
snmk	1	ip	割り当てる IP アドレスのサブネットマスク。デフォルトは ipad(IP アドレス) のクラスに応じた natural mask になる。
tmof	2	long	UTC からのオフセットを秒単位で示す。
rout	3	iplist	クライアントのいるサブネット上のルータの IP

			アドレスのリスト。
tmsv	4	iplist	time サーバ (RFC 868) の IP アドレスのリスト。
nmsv	5	iplist	name サーバ (IEN 116) の IP アドレスのリスト。
dnsv	6	iplist	DNS サーバ (RFC 1035) の IP アドレスのリスト。
lgsv	7	iplist	MIT-LCS UDP log サーバの IP アドレスのリスト。
cksv	8	iplist	Cookie サーバ (RFC 865) の IP アドレスのリスト。
lpsv	9	iplist	LPR サーバ (RFC 1179) の IP アドレスのリスト。
imsv	10	iplist	Imagen Impress サーバの IP アドレスのリスト。
rlsv	11	iplist	Resource Location サーバ (RFC 887) の IP アドレスのリスト。
hstn	12	str	クライアントのホスト名。
btsz	13	short	boot イメージの大きさ。
mdmp	14	str	クライアントがコアダンプする際のパス名。
dnsd	15	str	DNS ドメイン名。
swsv	16	ip	swap サーバの IP アドレス。
rpth	17	str	クライアントの root ディスクのパス名。
epth	18	str	Extensions Path (rfc2132 参照)
ipfd	19	bool	IP フォワーディングの許可。
nlsr	20	bool	non-local ソースルーティングの許可。
plcy	21	ippairs	non-local ソースルーティングの policy フィルタ。IP の終点とサブネットマスクの組のリスト。
mdgs	22	short	クライアントが再構成を行なえるように準備すべき IP データグラムの最大長。
ditl	23	octet	IP の TTL のデフォルト値
mtat	24	long	RFC 1191 に定義された Path MTU Discovery で使用する aging のタイムアウト値。
mtpt	25	mtpt	Path MTU Discovery で利用する MTU テーブル。
ifmt	26	short	Interface の MTU を指定する。
asnl	27	bool	クライアントの接続された IP ネットワークに含まれる全てのサブネットでは同じ MTU を使用するかどうかを示す。
brda	28	ip	クライアントの接続されたサブネットのブロードキャストアドレス。
mskd	29	bool	ICMP による subnet mask discovery を行なうか否かを示す。
msks	30	bool	ICMP による subnet mask の要求に答えるか否か。
rtrd	31	bool	RFC 1256 の router discovery を利用するか否か。
rtsl	32	ip	router solicitation 要求の宛先 IP アドレス。
strt	33	ippairs	static routing のための destination, router の IP アドレスの組のリスト。ここに default route を含めてはならない。
trlr	34	bool	ARP を使う時に、trailer の利用に関するネゴシエーションを行なうか否か (RFC 893)。
arpt	35	long	ARP キャッシュのタイムアウト。
encp	36	bool	Ethernet Encapsulation を利用するか否か。
dttl	37	octet	TCP の TTL のデフォルト値。
kain	38	long	クライアントの TCP が keep alive メッセージを送る間隔。
kagb	39	bool	クライアントが TCP の keep alive メッセージに garbage オクテットを付加するか否か。
nisd	40	str	NIS ドメイン名。

nisv	41	iplist	NIS サーバの IP アドレスのリスト。
ntsv	42	iplist	NTP サーバの IP アドレスのリスト。
nnsv	44	iplist	NetBIOS name サーバ (RFC 1001, 1002) のリスト。
ndsv	45	iplist	NetBIOS datagram distribution サーバ (RFC 1001, 1002) のリスト。
nbnt	46	octet	NetBIOS のノードタイプ (RFC 1001, 1002)。
nbsc	47	str	NetBIOS の scope(RFC 1001, 1002)。
xfsv	48	iplist	X Window システムのフォントサーバのリスト。
xdmn	49	iplist	X Window システムのディスプレイマネージャのリスト。
dht1	58	short	DHCP クライアントがユニキャストにより割当期限の延長を開始する時期を指定する。本実装では、サーバが返す割当期間に対する千分率で示す。
dht2	59	short	DHCP クライアントがブロードキャストにより割当期限の延長を開始する時期を指定する。本実装では、サーバが返す割当期間に対する千分率で示す。

システムコールって結構重いにょ

ikidou@mma.club.uec.ac.jp.

2001 年 4 月

Abstract

UNIX ではプログラム中からシステムを利用する場合システムコールを利用します。普段何気なく使うシステムコールですが、結構オーバヘッドがあるって知っていますか？これはそんなシステムコールのオーバヘッドを測定してみましょうっていう話でございます。

1 システムコールって？

OS はハードディスク、ネットワークインターフェイス等のハードウェア資源やプロセス情報、ユーザ情報等のソフトウェア資源 (二つ合わせて計算機資源と以後呼びます) を管理してます。

OS は管理している計算機資源を多重化、抽象化してユーザに提供します。提供する手段として UNIX ではシステムコールというインターフェイスを使用しています。システムコールを利用することでユーザはファイルがどういう種類のハードディスクにあるか等と気にする必要がありません。つまりシステムコールによってプログラムは高い移植性を持つのです。またインターフェイスという形で利用法を制限することで、システムが停止してしまうような操作をユーザに禁止することができます。つまり OS の安定性が高まるということです。

2 オーバヘッドがあるんですよ、奥さん

さて、ユーザにも OS にもうれしいシステムコールですが実は呼び出しにかかるオーバヘッドが結構でかいです。それを表す例 (?) [1]として標準入出力ライブラリのバッファリングが挙げられます。printf 等の標準入出力関数は呼び出し毎に read(2) や write(2) 等のシステムコールを呼ぶと効率が悪いため、データをバッファリングしています。

バッファリングされてる様子を見るため次のようなプログラムを実行してみましょう。

```
 1  #include <stdio.h>
 2  #include <unistd.h>
 3
 4  main(void)
 5  {
 6      while(1) {
 7          printf(".");
 8          usleep(100);
 9      }
10  }
```

実行すると'.' が連続で表示されずに、まとめて表示される[2]のがわかると思います。

ではちょっとバッファリングの影響を調べてみましょう。1Mbyte のデータをバッファのサイズを変えながら read(2) と write(2) してみます。測定マシンは Pentium 100MHz の FreeBSD-2.2.6[3]を使用しました。測定結果を図 1のグラフにしめします。

バッファサイズが小さいとシステムコールのオーバヘッドの影響がモロに出てるのがわかるかと思います。

3 getpid で測定しましょ

先の測定では正確なオーバヘッドがわかるわけではないので、もっと直接的にオーバヘッドを測定してみましょう。今度は getpid(2) システムコールをユーザ空間から処理する時間と getpid(2) がカーネル内で呼び出す getpid 関数の処理にかかる時間を測定し、その差を求めることでシステムコール呼び出しのオーバヘッドを算出します。それぞれの処理は 10 万回繰り返してそれにかかる時間を測定し平均を出してます。図 2がその結果です。結構差があります[4]よね。

[1]オーバヘッドの例というのは正しく無いっすね。その対策例ということで。
[2]バッファの大きさは 1Kbyte なので多分'.' が 1024 個ずつ表示されるでしょう。
[3]ふるっ!
[4]あくまでも "差" ですからね。全てのシステムコールが 15 倍も速くなるわけではないよ。

4　で、結論はな〜に

や、まあそのシステムコールにはオーバヘッドがあるよってだけです。

- カーネル内から getpid を呼び出す方法
- オーバヘッドが生じる理由
- オーバヘッドを減らすための方法
- 等々...

を書こうかと思ったのですが、巴里が燃えちゃったり、中間発表があったりするのでこれで終わりです。泣き事書くのはカッコ悪いけど、スマン。

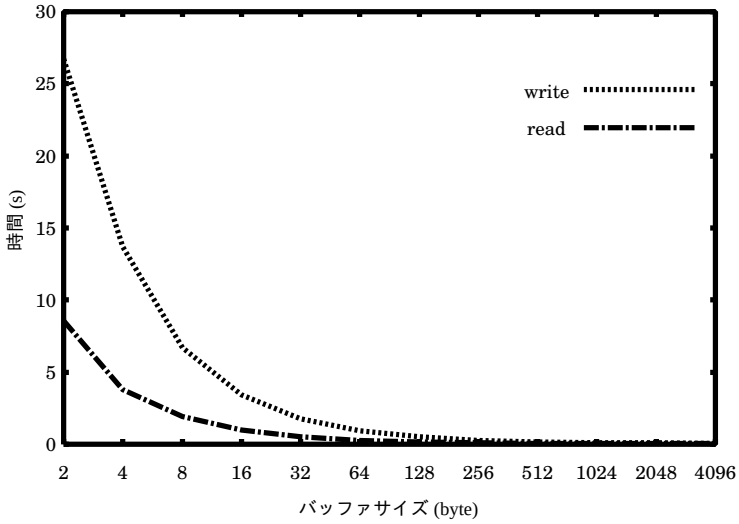

図 1: 入出力バッファの影響

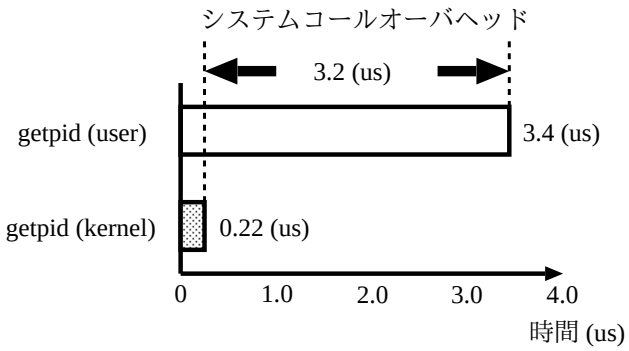

図 2: システムコールのオーバヘッド

書評

ikidou@mma.club.uec.ac.jp.

2001 年 4 月

Abstract

新年度も始まり学習欲が瞬間最大風速を記録している人も多い[1]と思います。この文章を読んでる人は何かしら計算機に関して興味をもっているのではないでしょうか? その興味を満たしてくれる情報は様々な方法で入手可能です。例えば、Web、マニュアル、詳しい人、プログラムソース、書籍等々があります。それぞれ長所短所がありますがここでは書籍に焦点を当て、個人的に面白いと感じている書籍を紹介します。

1 書籍の選び方

大学に入学し計算機関連の技術書を生まれて初めて購入したとき、値段の高さに驚いたものです。安くても 2k 前後、高いと 7k 前後もします。一冊買うと数日分の食費が無くなってしまうのは貧乏学生にはつらいですよね? 単純に値段が高いからといってその本がわかり易いというわけでもなく、高くてハズレの本を買った時は自己嫌悪してました。

そんなわけで書籍の選び方の自分ルールを書いときます。

- 興味や疑問をはっきりする

  一見あたり前なのですが「なんとなく必要そう」という理由で買っちゃう場合が結構ありました。そうした本は大抵読まれることなく「積読 (つんどく)」状態になってます。自分の中で疑問や興味を具体的にしてから、本を選ぶようにすれば良いでしょう。プログラムをしたこと無い人が「C 言語を勉強したいなぁ」と思って「有名だから」という理由で K&R の C 言語本を買ったら多分挫折する[2]でしょう。「UNIX 上の C 言語で Hello World プログラムを書きたい」等と自分の要求を具体的にしたほうが本の選択の助けになります。

- 立ち読みして内容を確認

  自分が本に求める物がはっきりしたら本を立ち読みしてみましょう。知りたいことが自然に頭に入ってくるような本があったら当たりです。買っちゃいましょう。もしなかったら無理して買わない方が良いです。

- 欲しい本は買う

  良い本というのは何かと参照する機会が多いです。その度、図書館で借りるのも一つの手段ですが、買った方が便利です。良い本を作った出版社、著者のためにも買っちゃいましょう。

2 読まない方が良い本

- 情報に嘘がある

  これは判別が難しいですが、運悪くこういう本にあった場合は野良犬に噛まれたと思って下さい。本に書いてある話はかならずしも真実では無いと常に頭にすみにいれておきましょう。

- 日本語が変

  文章が読みにくい本は内容が正しくても駄目です。苦労して読んで脳ミソが疲れた割には得るものが少ないと悲しくなります。ただ、知りたいことが書いてあるのがその読みにくい本だけであれば我慢するしか無いですけどね。

- 自分の知識が著者の対象としている前提知識に達していない

  これは本が駄目というわけではないですが、無理して読む必要はないでしょう。むしろその分野に拒絶反応がでる可能性があるので読まないのが吉です。成長してからのお楽しみということで。

参考文献

[1] 現実的な C プログラミング. Steve Oualline 著/岩谷宏訳. ソフトバンク.

C 言語の勉強をするために買った本です。初心者向けにわかりやすく書いてあります。文法の話だけではなく、上手にプログラムを組むためのプログラミング技術[3]も説明されてます。現在は第 3 版がオライリー・ジャパンより出版されています。

[1] 大抵ゴールデンウィーク後風力が急激に弱まります。
[2] K&R の本はプログラム経験のある人向けだと思います。
[3] 効果的なコメントの書き方、わかりやすい変数名、プログラムの設計法等々。

[2] エキスパート C プログラミング. Peter van der Linden 著/梅原系訳. アスキー.

C 言語でプログラムが書けるようになったら読んでみましょう。楽しみながら C 言語に対する理解を深めることができます。翻訳者によるこまかなツッコミが面白いです。

[3] UNIX システムプログラミング. 羽山博著. オーム社.

システムコールを使ったプログラムを組むことで、システム (UNIX) がどういう物かがわかります[4]。システムコールを使ったプログラムを解説する本では珍しく初心者向けなので、システムコール初めて (は〜と) という人にお薦めです。

[4] UNIX ネットワークプログラミング. 金内典充, 今安正和共著. オーム社.

[3] の姉妹本でネットワークプログラミングに特化した内容です。同じく初心者にお薦めです。

[5] 詳解 UNIX プログラミング [新装版]. W.Richard Stevens 著/大木敦雄訳. ピアソン・エデュケーション.

[3] よりも詳しくシステムプログラムについて書いてあります。今まで疑問だったことがこの本を読む事で次々解決しています。高いけどそれだけの価値はあるとおもいます。

[6] プログラミング作法. Brian W.Kernighan, Rob Pike 著/福崎俊博訳. アスキー.

上手なプログラムを組みたい人は必読です。私は洋書を読んでいたのですが、訳本が出たのでそっちを読みました。編集協力に harada さんの名が:)

[7] 珠玉のプログラミング. Jon Bentley 著/小林健一郎. ピアソン・エデュケーション.

[6] と同じで上手にプログラムを組みたい人向け。多少アルゴリズムよりかも。只今読んでる途中ですが目から鱗が落ちること度々です。

[8] 情報処理入門コース 2 オペレーティングシステム. 清水謙多郎著. 岩波書店.

OSって何やってるのか知りた〜いという人にお薦め。特定の OS に特化した内容ではないので、OS の概念を幅広く知ることができます。著者の清水先生は電通大で助教授をされていたこともあり、画面サンプルに uec の文字があったりします。

[9] UNIX カーネルの設計. Maurice J.Bach 著/坂本文, 多田好克, 村井純訳. 共立出版.

UNIX の内部構造を知りたい人向け。System V 系の話ですが BSD 系の人でもあまり気にならないと思います。アルゴリズムがソース形式で紹介されており読みやすいです。

[10] LINUX デバイスドライバ. Alessandro Rubini 著/山崎康宏, 山崎邦子訳. オライリー・ジャパン.

カーネルプログラムしたいけど良くわからないという人向け。Linux のカーネル 2.0 系ですけど参考になるでしょう。ソース例が多く実際に動作させながら読むのがお薦めです。

[4] プロセスが何かとかファイルが何かとか...

i-mode 絵文字って mail の Subject で送れるの？

uirou@mma.club.uec.ac.jp.

2001 年 4 月

Abstract

ある日、ある時、ある時間。僕は、

i-mode 絵文字って mail の Subject で送れるの？

との質問を承った。えーと、うーんと、えーと、うーん…

わかりません。

…えーと、これではいかんので、調べてみることにした。これが、深遠なる ISO-2022 への道だとは知らずに…

1 i-mode 絵文字ってなんじゃらほい

まずは、i-mode 絵文字とは何か、を調べなければならない。i-mode 絵文字。これは、

SJIS の外字領域と呼ばれる所の、`0xf89f`〜`0xf8fc`、及び `0xf940`〜`0xf9af` までの範囲に定義された文字の事である。

ということらしい。へー。そーなんだー。すごいねー。…で？SJIS が mail の Subject に書けるかって？書けるわけないのでは。よーわからんけど mail の Subject に SJIS は書いちゃいけないって話だよ。うん。…では、仕事にならないので[1]、それでも mail の Subject に書く方法を調べるわけである。

2 つか、Subject に日本語ってつかえるのでは？

で、よーく考えると、Subject に日本語って使えるよね。で、その時って必ず`=?ISO-2022-JP?B?...?=`ってあんじゃん。あれなんでは。ははぁ。なるほど。なるほど。ISO-2022-JP すか。

SJIS じゃないじゃん！

てことはあれですか。SJIS から ISO-2022-JP に変換しなきゃいけないってことですね。ISO-2022-JPってあれだよね。JIS の文書って奴。あれって、`ESC$(B` とかいう感じのエスケープシーケンスみたいな奴を使って、漢字にしたり、ASCII にしたりする奴だよね。ふむ。んじゃぁ、まずは、そういう変換を作らないといけないわけね。

3 SJIS to ISO-2022-JP

ということで、jcode.pl[2]を読んでみた。どうも、SJIS には ASCII と半角カナ、それから漢字があるらしく、ASCII と半角カナは 1byte、漢字は 2bytes で表現されているみたい。で、ASCII の時には `ESC(B`、半角カナの時は `ESC(I`、漢字の時には `ESC$B` を使うらしい。で、半角カナは 8 進数で 0241〜0337 を 0041〜0137 へ、よーするに、8bit 目 をおろせばいいのね。漢字は…なんか一度 EUC に変換してから、JIS に変換してるみたいだ。まぁ、いいや。とりあえず機械的に変換すればいいらしい。ごにょごにょっとやって実装完了。[3]んじゃぁ、あとは`=?ISO-2022-JP?B?...?=`の秘密を解析すれば大丈夫だね。

4 `=?ISO-2022-JP?B?...?=`

この形式って、確か、`=?`で始まって、`?`で区切られて、最初のフィールドが文字コード、次のフィールドが encode の形式 (B は base64)、最後のところに encode された文字列が入るんだよね。と、いうことで、もう JIS に変換はできたわけだから、あとは base64 encode をすればいいわけだ。かんたんかんたーん。base64ってあれでしょ。`A-Za-z0-9`、そいから+と`-`で 24bit づつ表して、4byte で 3byte を表すって奴でしょ。知ってるしってるー。[4]と、いうことで、てけとーに base64 encode して日本語 subject を作ってみました。これを、i-mode の端末に向けて mail してみると…

出ました！ちゃんと i-mode 絵文字が出ましたよ。すごい。わーい。できちゃったよ。うひょひょ。

[1] 仕事だったらしい
[2] ftp://ftp.iij.ad.jp/pub/IIJ/dist/utashiro/perl/
[3] ちなみに、http://alfin.mine.utsunomiya-u.ac.jp/ niy/algo/ に変換公式がわかりやすく載っている
[4] しったかーには気をつけましょう。こういう手合いで本当に知っている奴はいません

5　でも、まてよ。本当に絵文字は送ってもいいんかい？

さて、ここで問題です。あなたは、

電子メールで半角カナの Subject は送ってはいけない

というお話を聞いた事がないでしょうか。僕は聞いたことがあります。と、いうことで、実験してみました。さいわい、僕の作った program は、半角カナを全角カナに変換するような処理は入っていない[5]ので、SJIS で半角カナを書いて送れば、そのまま base64 encode されて mail の Subject になるはずです。と、いうことで、やってみました。

すると、どうでしょう。送れてしまうし、表示できてしまうじゃないですか。はて。なんででしょう。そういうもんなんでしょうか。僕の記憶間違いって事なんでしょうか。いやいや、そんなことは無いはずです。少しくらい自分の記憶に自信を持ったっていいはずです。と、いうことで、検証してみます。

mail のヘッダ部は 8bit 目が立っているものは送ってはいけない
　ある意味真。RFC822 では、8bit 目が立っているものについては、現れないことになっている。

でも、base64 encode された文書は 8bit 目は立たないでは
　これも真。確かに、base64 encode された文書はどうやっても `A-Za-z0-9+-`しか現れないので、rfc822 的には大丈夫。

ていうか、JIS の文書って 8bit 目は立たないんじゃないの？
　それが、なんか知らないけれど SJIS の i-mode 絵文字を JIS に変換すると、8bit 目が立つのよね。だとすると、encode しなきゃいけないんじゃないかなぁ。

でも、JIS の文書でいくら 8bit 目が立ったとしても、base64 encode してるんだから、いいんだよね？
　いい…と思うけど。

あれれ。問題ない感じです。じゃぁ、どこで半角カナがだめだなんていうお話になったんでしょう。

6　答えはいつも規格書等で文書化されている

で、ちょっと、待ってください。mail のヘッダ的には問題ないってことは、JIS のコード自体で半角カナはだめだということにならないでしょうか。って、でも、SJIS の半角カナを JIS の半角カナへ変換する公式も存在するわけですし、それが無いってことはないでしょう。ちょっと調べてみます。

で、どうも、日本の工業規格って奴の JIS の中に、JISX0201 というので、ASCII とカナの文字集合が定義されていて、JISX0208 というので、漢字の第一水準と第二水準の文字集合が定義されているらしいです。で、この文字集合って奴がなんつーかいまいちよくわからない概念で、どうも、文字の字面だけを決めているらしいんです。と、いうのも、JISX0201 で定義されている (ASCII と) カナなんですが、同じカタカナが JISX0208 でも定義されているわけなんです。で、これっていうのは、別に JISX020?でこっちが半角カナであっちが全角カナだ、なんて、決まってないというのです。はーはー。なんか、怪しい歴史の匂いがしてきました。なんだか面倒くさそうなので無視して先に進む事にします。で、jcode.pl によれば、その JISX0201(ASCII とカナの方) のカナは、半角カナで、JISX0208(漢字の方) のカナは、全角カナということになるようです。確かに、JISX0201 のカナは 1byte で表現されていて、JISX0208 のカナは 2bytes で表現されているわけで、なんとなく半角カナ、全角カナというのもうなずけます。

さて、それでは、JIS の文書ってのは何をさしているんでしょう。と、いうか、JISX0201 や JISX0201 には、エスケープシーケンスのお話は全く出てきませんでした。じゃぁ、そのエスケープシーケンスはどこで定義されているんでしょう。うーん、よくわかりません。

7　iso-2022 に答えあり

で、行き着いたのが iso-2022 です。この規格はなにやらむつかしい概念がふんだんに盛り込まれてはいるのですが、まとめて言うと、JISX0201 や JISX0208 等で定義されていたりする文字集合を格納する箱を 4 つ用意して、エスケープシーケンスを用いてそれらの箱の中身を入れ換えたり、現在使用中の箱を変えたりするという仕組みです。要するに、現在使用中の文字集合という概念があって、その現在使用中の文字集合をエスケープシーケンスで変えることができる、というわけです。たとえば、普段は JISX0201 の文字集合を使っていて、`ESC$B` というエスケープシーケンスを用いると、JISX0208 の文字集合に変わるというわけです。

おっと、これって、要するに、今まで JIS の文書というやつでは無いでしょうか。きっとそのとおりです。でも、何処にも JIS の文書であるなんて書いてありません。というか、JIS 規格にはこのようなエスケープシーケンスが定義されているとは書かれていません。

[5] 人はそれを怠慢という

はて。てことは、JIS の文書っていうのは、そもそも存在しなくて、ISO-2022 で書かれた日本語の文書の事が俗に言う JIS の文書ってことなんでしょうか。たぶん、そうです。きっと、そうです。

さて、ISO-2022 という規格はいろいろ出来てしまうある意味万能な規格なのですが、いろいろな事が出来てしまうがゆえに、ISO-2022 を完全にサポートするような物を作るのは大変です。そこで、RFC1468 で ISO-2022-JP という ISO-2022 のサブセットとなる規格が規定されています。これで定義されているエスケープシーケンスはたったの 4 つで、とても簡単です。ところが、JISX0201 については、ローマ字セットのみということになっています。って、それって、半角カナは ISO-2022-JP ではないという事ではないでしょうか。でも、逆に、JISX0208 については、制限がありません。すると、ISO-2022-JP においては、半角カナは送ってはいけないが、JISX0208 はすべて送っても良い。ただ、i-mode 絵文字に関しては、SJIS の外字領域に定義されているわけで、SJIS の外字領域は、JISX0208 の未定義領域を使っているわけで、その未定義領域に関する事は ISO-2022-JP では定義されていないわけで、結局、

半角カナは送ってはいけない、i-mode 絵文字については送っていいのかどうかはわからない

という事になります。

8 旅の終わり

やっと答えにたどりつきました。しかし、求めていた答えではないようです。それでも、問題なく送れてしまうのではあるので、「まあ、いっかぁ」というお話にもなるわけですが、よい子の皆さんはこういうことをまねしてはいけませんよ。

参考文献

[1] RFC1468,ftp://ftp.uec.ac.jp/pub/doc/RFC/rfc1468.txt

[2] RFC1554,ftp://ftp.uec.ac.jp/pub/doc/RFC/rfc1554.txt

[3] ISO,http://www.iso.ch/

マルチスレッドのおはなし

ottan@mma.club.uec.ac.jp.

2001 年 4 月

1 まえふり

ここ最近で、一番邪悪で面白いネタだと思ったのは、Java アッセンブラーです。
このネタは “Java のバイトコードはアセンブラのニーモニックに相当するようなモノで構成されているのではないか” という仮定から来ています。
“Write once, run anyware.TM” はそのままに、オブジェクト指向なんてタルいものは捨てて、コンピュータに優しく人間に厳しいプログラミングに回帰してみませんか？(^^
というわけで、今回のネタは、前回の “セルオートマトンのおはなし” のセルオートマトン法でもって動物の体表面の紋様を再現するってやつを、デュアルやクアッドな SMP マシンを用意してマルチスレッド なプログラム で動かしてみようというお話です。
ま、ようするに、たくさん CPU ブンまわしてシマシマを描こうということですな。

- マルチスレッドの基本と実装
- セルオートマトンのおさらい
- serial program
- simple threaded program
- sync threaded program
- タスクの粒度とパフォーマンス

と言うような流れで話を進めていきます。

2 マルチスレッドってなに？

マルチスレッドは、並列処理を行うための 1 つの手段です。
一般に、マルチタスク OS では CPU、メモリ、ディスク、ネットワークなどの資源の割り当てや管理をプロセス[1]単位で行っています。
マルチスレッド (の利用できる) 環境では、プロセス単位による管理より柔軟性を持たせるために、プロセスの中にもう 1 つ小さなタスクの単位としてスレッドを用意し、CPU など一部の資源を個々のスレッドに割り当てます。
そこで、複数のスレッドを協調動作させることによって並列処理を行うのが、マルチスレッドプログラミングです。
もちろん、複数のプロセスを協調動作させるマルチプロセスによる並列処理も可能です。しかし、

- 新しいプロセスの生成はコストが高い
- プロセスどうしは完全に独立しているため、データのやり取りをするには別の手段を用意しなければいけないので、またこれにもコストがかかる

など、新たなタスクの生成やデータの共有のオーバーヘッドが大きいくなってしまいます。あと、何が嫌って、プログラムするのがめんどくさいんだもん（ ^^;
今回の主な目的である複数の CPU 資源の有効活用の他にマルチスレッドの利点としては、

- ブロッキング処理によるプロセス停止の回避
- 非同期イベントの処理
- リアルタイムスケジューリング

などがありますが、この辺は参考文献読んでください。
で、実際にマルチスレッドプログラミングするためには、スレッドを制御するためのライブラリ関数を利用します。
ここでは、UNIX 環境下で標準的な POSIX thread (Pthread) を取り上げます。

[1] ここでは、実行状態にある個々のプログラムをプロセスと呼びます。例えば、元は同じプログラムでも複数実行されていれば、それぞれ別のプロセスです。

3　スレッドライブラリの実装

さて、スレッドライブラリを使いさえすれば複数の CPU をブンまわせるかというと必ずしもそうではありません。スレッドライブラリの実装のしかたに依存します。

この辺の話を新入生にもわかりやすく書くってのは手にあまるので、難しかったらこの章は読み飛ばしてかまいません。

マルチスレッドの実装のしかたには、カーネル空間で実装、ユーザー空間で実装、その２つをハイブリッドして実装の３つがあります。

それぞれ長所短所があります。

1　カーネルレベルスレッド

カーネル空間内に実装されたスレッドです。カーネルが直接スレッドの面倒を見るので、それぞれのスレッドにまで CPU 資源を割当てることができます。しかし、すべてカーネルが処理しなければいけないので、スレッドの生成やスレッド間のコンテキストスイッチの処理など重くなってしまいます。

非同期処理などの CPU 集約型ではないのタスクが多いプログラムや大量のスレッドを生成したりするプログラムは苦手です。

手近な実装例としては、LinuxThread があります。

カーネルレベル / 生成が重たい / CPU 資源も利用可能 / LinuxThread

2　ユーザーレベルスレッド

ユーザー空間に実装されたスレッドです。すべてユーザーランドで動作するため動作が軽く、実装が簡単です。

ただし、カーネルからはプロセスしか見えておらず、個々のスレッドはカーネルレベルでのスケジューリングとは別に、プロセス内でユーザーレベルでスケジューリングが行われるため、CPU 集約型のタスクは他のプロセスの負荷の影響を受けてしまいます。また、カーネルからは個々のスレッドは見えてないので、CPU ごとにスレッドを割り振ったりもできません。よって、今回の目的にはちょっと向きません。

実装例としては、MIT (proven) thread, PTL などがあります。*BSD の pthread は MIT のものをベースにしているようです。

ユーザーランド / 軽い / CPU 資源はもらえない / MIT thread, PTL

3　2 レベルスレッド

カーネル空間とユーザー空間両方にまたがって実装されたスレッドです。カーネルから見えるプロセスやライトウェイトプロセスと個々のスレッドの対応が柔軟にできるため、スレッドのタスクの性質にあわせてカーネルから CPU 資源をもらったり、ユーザーレベルで動作させたりできます。

いいことづくめのようですが、スレッドライブラリを実装する方は大変です。ですので、いまのところ Solaris や Tru64 UNIX などの商用 UNIX でしか利用できません。

２レベル / いいとこどりのようだけど、実装が大変 / Solaris, Tru64UNIX

4　おあとがよろしいようで

おはなしの途中ですが、ていうか、全然予定を消化していないのですが、締め切りすぎちゃってるので今回はここまでです。調布祭版に掲載されるかも知れない続編をお待ち下さい。

4　参考文献

Bradford Nichols, Dick Buttlar, Jacqueline Proulx Farrell 共著,
榊 正憲 訳,
Pthreads プログラミング,
オライリー・ジャパン

ottan@mma.club.uec.ac.jp,
セルオートマトンのおはなし,
MMA 百萬石 2000 年 新歓,
http://www.mma.gr.jp/

IP over e-mail

shimiz98@mma.club.uec.ac.jp.

2001 年 4 月

Abstract

以前の百萬石に uirou さんが ssh over HTTP の記事を書きました。とちゅうにファーやーウォールなどがあって、直接 TCP のコネクションを張れないホスト間でなんとか TCP のコネクションを張るために、TCP ストリームのデータを HTTP Proxy サーバを介して送ってしまおうと言うものでした。ssh と名前にはついていますが、TCP のコネクションなら他のものでも使えるはずです。

HTTP の上で出来るなら、Web と並んでインターネットでよく使われるサービスであるメールでもできるといいな、と言うわけで、IP over e-mail「milk」 の登場です。

1 どうしてトンネルデバイスを使うことにしたのか

しかし TCP のポート転送だと ping は通らないし、似たことをやってもつまりません。なによりコネクション指向の TCP のデータをコネクションレスのメールとして送るにはコネクションの確立、切断、再送、タイムアウトとめんどうな事が多いのです。結局 TCP がやっていることと同じ事をプログラムしないといけないのです。これは無駄と言うものです。

そこでトンネルデバイスを使って、IPv4 のパケットを全部メールで送ることにしました。トンネルデバイスというのは仮想的なネットワークインターフェースです。あたかもネットワークインターフェースの様に見えて、ifconfig でアドレスを割り当てたり、route コマンドでルーティング情報を設定できたりできますが、ユーザランドプログラムからキャラクタデバイスファイルとして read するとそのインターフェースに届いた IP パケットが読み込めて、write すると IP パケットが送り出せるというものです。こうすれば面倒なことは全部 TCP に任せられます。途中でメールが消えようが、順番が狂って届こうがその辺は TCP がうまく処理してくれます。

2 どうやってうごくのか

自宅のマシンが 2 台とも NetBSD なので NetBSD 用に書くことにしました。NetBSD ではトンネルデバイスは /dev/tunN(N は数字) です。ifconfig でアドレスを割り当てて、route コマンドでルーティングしておいて、ping を打ちながら /dev/tun0 を read してみると IPv4 のバージョン番号 「4」から始まるデータが読み込めます。これを 16 進ダンプして To: と From: をつけて sendmail の標準入力から流し込むこんでメールをおくります。

メールを受け取る方については、メールが届いたらプログラムを起動してその標準入力にメールを流し込むという設定ができるので、この機能を使います。トンネルデバイスは既に open されていると他のプロセスが open しようとしても失敗するので、メールが届いたら起動されるプログラムから直接トンネルデバイスに書き込むわけにはいかないので、いったん標準入力から入ってきたデータを Unix ドメインソケットに書き込んで、その Unix ドメインソケットに書き込まれたメールをトンネルデバイスを開いているデーモンプロセスで 16 進ダンプからバイナリデータに戻してトンネルデバイスに write すると完了です。

3 どうなったか

K6-2 300MHz で 100Base-TX のマシンから 486 66MHz の 10Base-T のマシンへ ping を打ったところ RTT は 3.5 秒くらいになりました。

```
% ping -i 3 kkana
PING kkana.myhome (192.168.200.2): 56 data bytes
64 bytes from 192.168.200.2: icmp_seq=0 ttl=255 time=4249.275 ms
64 bytes from 192.168.200.2: icmp_seq=1 ttl=255 time=3610.617 ms
64 bytes from 192.168.200.2: icmp_seq=2 ttl=255 time=3473.030 ms
64 bytes from 192.168.200.2: icmp_seq=3 ttl=255 time=3468.757 ms
64 bytes from 192.168.200.2: icmp_seq=4 ttl=255 time=3440.743 ms
64 bytes from 192.168.200.2: icmp_seq=5 ttl=255 time=3778.143 ms
64 bytes from 192.168.200.2: icmp_seq=6 ttl=255 time=3359.663 ms
64 bytes from 192.168.200.2: icmp_seq=7 ttl=255 time=3386.090 ms
64 bytes from 192.168.200.2: icmp_seq=8 ttl=255 time=4069.437 ms
^C64 bytes from 192.168.200.2: icmp_seq=9 ttl=255 time=3246.063 ms

----kkana.myhome PING Statistics----
10 packets transmitted, 10 packets received, 0.0% packet loss
```

```
round-trip min/avg/max/stddev = 3246.063/3608.182/4249.275/326.702 ms
```

おなじ構成で ssh してみました。パスワードプロンプトまで 30 秒、一文字入力してエコーバックされるのに 5 秒ほどかかりますが、ちゃんと動きます。

```
% ssh kkana
shimiz98@kkana.myhome's password:
Last login: Fri Apr  6 02:00:48 2001 from mmegu.myhome

NetBSD 1.5 (KANA) #2: Mon Feb 19 07:30:33 JST 2001

Welcome to NetBSD!

kana% netstat -i -I tun0
Name  Mtu   Network       Address              Ipkts Ierrs    Opkts Oerrs Colls
tun0  1500  <Link>                               126     0      119     0     0
tun0  1500  192.168.200   192.168.200.2         126     0      119     0     0
```

4 もっとよくするには

16 進ダンプは人には読みやすいが、データが多くなるので Base64 などにするつもりです。Unix ドメインソケットを使って親子関係のないプロセス間でファイルディスクリプタを受け渡せるそうなので、メールが届いたら起動するプログラムはデーモンプロセスに標準入力のファイルディスクリプタを渡すようにして無駄なメモリのコピーを減らすつもりです。送信と受信を同時にできるようにすると、高速化しそうな気もしますが、変わらない気もします。SMTP を自分でしゃべれば高速化できそうです。

5 セキュリティはどうなっているのか

何もチェックしていないので IP アドレスを偽ったパケットなどもつるつるです。せめて認証はしたいところですが、面倒です。ここは他力本願でいきましょう。と言うわけで次回は milk + IPsec です。

参考文献

[1] UNIX ネットワーキングプログラミング 第 2 版 Vol.1 W.Richard Stevens トッパン

[2] BSD Magazine No4.2000 プロジェクト短信 PPxP 真鍋 敬士 アスキー

[3] Unix Magazine 95 年 11 月号 UNIX 流プログラミング 小泉 貴史 アスキー

[4] 百萬石 1998 年秋号 ssh over HTTP uirou MMA

SDL でゲームを作ろう
～接触編～

fujita@mma.club.uec.ac.jp.

2001 年 4 月

Abstract

SDL(Simple DirectMedia Layer)[1]とは、BSD や Linux,Windows,MacOS,BeOS などで容易にゲームを開発するためのライブラリです。SDL は各 OS のナイティブなインターフェースを一つの API に統一します。これによって可搬性のあるソースコードを書くことができるようになります。本記事ではゲームの流れに沿って SDL の使用方法を解説し、実際に簡単なゲームを作成します。

1 はじめに

どうもこんにちは。fujita@無職生活真最中です。

さて皆さん、UNIX 系 OS でゲームというと何を思い浮かべるでしょうか。Nethack だったり nethack だったり nethack だったり、まあ様々でしょう。Nethack は御存知の通りターミナル上で文字を動かしてゲームを行なうという見た目地味なゲームですが、X Window System を使ったグラフィカルなゲームも多数存在しています。これらのゲームではグラフィックを用いることによって情報量を増やしたり、見た目の派手さでプレーヤにインパクトを与えています。

しかし X Window System を用いてゲームを開発するときに問題となるのはライブラリが煩雑であるということです。現実逃避として片手間にプログラムをするにしても、X Window System では作業が複雑化してしまい本末転倒です。素の Xlib ではなく、GTK などのツールキットを使えば多少の負荷は軽減されるでしょう。しかし BGM を鳴らしたり CD-ROM からデータを読み込んだりと、本格的にゲームを作ろうとしたときにはツールキットだけでは力不足です。

さらに UNIX 系 OS 上で動くゲームを他の OS 上で動かそうとする場合にも、ソースコードの可搬性という問題が起こります。BSD や Linux では X Window System を使い、Windows では Win32 の API や Direct X を使い、といった風にプログラムを書くことも出来ますが、ifdef の嵐に巻き込まれて遭難しそうです。

そこで上記の問題を解決するために SDL(Simple DirectMedia Layer) を用います。SDL はゲームを簡単にプログラミングするためのライブラリです。SDL はゲームに必要な画像描画、BGM 演奏などといった API を提供します。SDL では OS 固有のインターフェースをライブラリレベルで吸収した上で API を提供しているので、一つのソースコードによって様々な OS 上で動くプログラムを作ることが出来ます。

例えば CCLUB 別働隊[2]が作成した「えここ de ふぁいと！SDL 版」は SDL を使用したゲームの一例です[3]。

[1]http://www.libsdl.org/
[2]http://cclub-flying.dsl.gr.jp/
[3]要するに、筆者はこういうダメなモノを作りたかったらしい

2　SDL の機能

SDL は画像描画、イベントハンドリング、オーディオ操作、CD-ROM 操作、マルチスレッドなどの機能を有します。全ての機能を説明すると複雑になって逆に理解の妨げになるので、本記事では必要最低限と思われるごく少数の機能を説明します[4]。

以下ではゲームのプログラムとしての流れを追っていきながら、対応する SDL の機能を解説していきます。まずゲームが始まると環境を初期化します。そしてゲーム本体の中に入っていきます。ゲーム本体では、ユーザの操作を認識してそれに合った処理を行い、ゲームの画面を描き換えるという作業の繰り返しとなります。ゲーム本体の中でゲームの終了条件を満たしたら、後片付けをしてゲームを終了します。

1　初期化

まず最初にやっておかなければならないのは SDL_Init() です。SDL_Init(SDL_INIT_VIDEO) とすればグラフィックが初期化されて使用可能になり、SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) とすればグラフィックとオーディオが使用可能になります。

次に SDL_SetVideoMode() で描画用の surface を作ります。Surface とは X Window System でいうところの Drawable です[5]。Surface に様々な処理を加えることによって、実際のゲームのウィンドウに描画することが出来ます。Surface の特定の矩形領域を塗り潰したり、surface 同士を重ね合わせることも可能です。

さて、ゲームで重要なのは画面を蠢きまわるキャラクタです。自機だったり敵機だったり、シューティングだったら弾丸、ファイヤーボール、十字架、オイル缶、その他いっぱいあります。これらのキャラクタはスプライトとも呼ばれ、画面上に逐次描画されてゲームの動きを表現します。即ち、先に作った大元の surface とキャラクタ群の surface を重ね合わせたものがゲーム画面となります。キャラクタの surface を得るには元の画像を読み込む必要があります。現在 SDL が対応しているのは Windows Bitmap だけのようで、SDL_LoadBMP() で bitmap ファイルを読み、surface を得ることが出来ます。

SDL_SetColorKey() を用いると surface を重ねるときの透過色を指定することが出来ます。キャラクタ同士が重なり合っても、前景のキャラクタによって背後のキャラクタが塗り潰されてしまうことを防げます。また、SDL_DisplayFormat() で surface をハードウェアに適した状態に作り換えることが出来ます。これによって透過色や透明度を指定して surface を重ねるときに、ハードウェアの機能を利用して高速な処理を行うことが可能です。

あとはキャラクタの座標や状態といったパラメタを初期化するだけです。キャラクタのデータ構造は各人のセンスで作ってください[6]。

2　イベントとパラメタの処理

ユーザのキーボードやマウス操作はイベントとしてプログラムに伝えられます。SDL では SDL_PollEvent() で存在しているイベントを得ることが出来ます。得られたイベントの種類に応じてキャラクタのパラメタを書き換えます。例えば矢印キーが押されたら自機の座標をキーの方向に対応するようにアップデートします。プレーヤが直接操作しない、敵機などのキャラクタはイベントとは関係なくパラメタを変更する必要があります。この辺のキャラクタの動作、当たり判定などのアルゴリズムも好みでやってください[7]。

3　キャラクタの描画

キャラクタの座標が決定したらキャラクタの画像を描画します。ある surface を他の surface に重ね合わせるには、SDL_BlitSurface() を使用します。重ね合わせた部分は SDL_UpdateRects() で実際に描画します。動くキャラクタを描画するときには直前に描画した画像を消去しておきます[8]。

4　終了

ゲームを終わるときには SDL_FreeSurface() によって作成した surface を解放します。また、SDL で使用されているシステムを SDL_Quit() で閉じ、exit(3) します。

3　まとめ

SDL を用いた単純なプログラムの例として単一画面シューティングもどきを http://delegate.uec.ac.jp:8081/club/mma/%7Efujita/ に示します[9]。SDL-1.1.x およびスプライト用の bitmap ファイルが別途必要です。

[4] BGM とかスレッドとかはメンドいのでやりたくない、というのが本音らしい
[5] PhotoShop でいうところのレイヤ、つーか絵が描かれた透明なシートがいっぱい重なってると思ってください
[6] キャラクタの XY 座標、状態 (右に移動中とか攻撃中とか)、surface へのポインタくらいが典型だと思います
[7] つーか、ここが一番ムズいんですけどね
[8] エラくあっさりした節ですが、他に書くことがないのだからしょうがない
[9] 所詮素人が作ったプログラムなので、状態遷移とかのアルゴリズムの不備は気にしないでください…

本記事では少ないSDLのAPIだけでゲームを容易に作成出来ることを紹介しました。あとはイベント処理やキャラクタのパラメタ処理を変えることによってシューティングやアクション、パズルなど様々なゲームを作ることが出来るでしょう。では次回「SDLでゲームを作ろう ～発動編～」でお会いしましょう[10]。

[10]多分ウソです。でもスクロール型アクションゲームを作りたかったりします

あるひとりコント

fujita@mma.club.uec.ac.jp.

2001 年 4 月

——開幕

生協二階の学食、対面してテーブルに座っている学生二人。テーブルの上にはレポート用紙やグラフ用紙、電卓が散乱している。どうやら実験レポートを書いているよう。

学生 A:「新番組は見た?」

学生 B:「一応」

A:「で、どうよ」

B:「また深夜枠が増えたね。あと、なぜかわからんが日曜放送のヤツがいっぱい」

A:「ふむ。何か気になったのはあった?」

B:「『探偵少年カゲマン[1]』は気になるねぇ。今のガキで山根あおおにを知ってるヤツはいるんか? 大学生でも怪しいぞ。そんなもんをアニメ化しようとする NHK 教育の気が知れん」

A:「そ、そう……」

A:「原稿のネタになりそうなのはあったかね?」

B:「うーん、そうだな。『daemonic・レイヤー[2]』なんてのはどうだろう」

A:「ほう」

B:「root と呼ばれる操縦者が daemon を操って互いに戦わせるのだ。apache とか squid とか bind とか。sendmail 対 qmail だとやっぱり sendmail の勝ちかな」

A:「……イマイチ絵づらを想像しにくいんだが、どうやって戦うんだ?」

B:「お互いに daemon 立ち上げて、アタック大会」

A:「うわ、地味!」

A:「もーちっと華やかなのはないのか? いくら何でも黙々とアタックするだけのアニメはイカンだろう」

B:「そう? じゃ『クラッカー・プリンス[3]』とか」

A:「……プリンス? プリンセスじゃなくて?」

B:「主人公は某プロバイダの管理者で、ある日 12 人のクラッカーに自分のサーバをアタックされる。ソフトをバージョンアップしても OS インストールし直しても執拗にアタックしてくるクラッカー達。root 権限を奪ったクラッカー達はそこを踏台にして、さらにアタックをしかける。山のように届く警告メール、警察の介入、果ては FBI まで登場」

A:「シャレにならんぞ、それ」

B:「主人公とクラッカー達との攻防を描く、愛と感動の一大巨編だ」

A:「で、結局最後にはどうなるの?」

B:「そりゃぁ12 人のうちの一人とねんごろになるのが最終目標さね」

A:「そこだけギャルゲーしてどーする! しかも男同士で!!」

A:「何が悲しくて野郎をくどかないかんのだ。おなごを出せ、おなごを。キャッチーなおなごを出しときゃ世の中何とかなる」

B:「うーむ。『そふとパーティー[4]』なんかは、全面に押し出してるわけじゃないが、一応女子が出てくるけど」

A:「……や、何かタイトルだけで内容が想像出来そうな気が」

B:「なら話は早い。これは主人公が同人ソフト界でのし上がっていく様を空前のリアリティをもって描く、今までにない新しい作品だ」

A:「どうせギャルゲーのキャラクタを使った格ゲーを作ったり、CG 集出したり、完全オリジナルノベルゲームを作ったりするんだろ……?」

B:「わかってるねぇ。ただ、規制の厳しい会社のクローンゲームを作ったりしてると危険だぞ。警告されて、あわや訴えられる直前までいくからな。そうなると地下に潜るしかない」

A:「コメントを控えさせて頂きます」

B:「あと、既成ソフトに不満があるからといって自分で似たようなものを作るのは良いけど、必要以上にケンカ腰になったり大元の名称の一部を使用するのはやめといたほうがいいな。人気絶頂なのに頒布停止処分はイタイだろ」

A:「そのくらいにしとけ。いや、マジで」

---

[1]探偵少年カゲマン、NHK 教育「天才テレビくん」内。原作は小学館の「名探偵カゲマン」。15 年以上前のマンガである

[2]エンジェリック・レイヤー、テレビ東京。CLAMP 原作のマンガのアニメ化。少年少女がエンジェルという人形を操作して戦う、典型的な対戦マンガ

[3]シスター・プリンセス、テレビ東京。雑誌の誌上企画をアニメ化したもの。突然主人公に 12 人の妹ができるという、すさまじい電波な内容で今後が期待される

[4]こみっくパーティー、UHF 各局。Leaf の同名ゲームをアニメ化したもの。すっごいナンパな大同人物語、というとやはり語弊があるだろうか

A:「うーん、どれもいまいちネタとしてはインパクトに欠けるな」

B:「なら魔法少女モノなんてどうだ?」
A:「何だ、そういうのもあるんじゃないか」
B:「主人公はBSD国の王女で、行方不明になったLinux国の王子を探しにこっちの世界にやってくるわけだ。色んな事件を魔法で解決しつつ、王子を探すというわけ。魔法少女モノなのでそれ系の小物も用意して、お共の生き物には三叉を持ったdaemonってところかな。あと、ライバル役の魔法少女でSYSV国の王女も登場」
A:「……で、タイトルは?」
B:「ずばり『とこちゃんさん☆[5]』で」
A:「『マジカルとこちゃん[6]』のパクリじゃねーか!!」

暗転。閉幕——[7]

[5]コメットさん☆、テレビ東京。過去にも放映されて人気を博した作品のリニューアル版。大きなお兄さんには受けそうだが、問題はお子さまに受けるかどうかだ
[6]数年前に怒涛の勢いで誕生した架空企画。詳しくはhttp://TokoChan.hauN.org/
[7]内輪受けにもなってない、こんなネタを載せて良いものだろうか?

編集後記

abe@mma.club.uec.ac.jp.

2001 年 4 月

毎度の事ですが、今回の百萬石編集はそのなかでもさらにいっぱいいっぱいでした。百萬石的にもいっぱいっぱいで僕自身もいっぱいっぱいだった気がします。一番楽だったのは去年の春、つまり始めて百萬石の編集担当をした時です。みなさん〆切前に提出していただけましたし、編集的にはあとは編集するだけ状態が 1 週間以上もありました。それからすると今回の壮絶さは半端ではないとおもわれます。まず事前に発表した編集予定によると

- tex 以外のデータ〆切　3/25
- tex の〆切　3/28
- マスター製作予定　3/31
- 印刷予定　4/1 or 2
- 新歓　4/6,9,10

のはずだったのですが、実態は

- tex 以外のデータ〆切　4/2
- tex の〆切　4/6
- マスター製作予定　4/6
- 印刷予定　4/6 20:00
- 新歓　4/6,9,10

になってしまいました。
今回の敗因は

1. せっぱつまった感を匂わせたメールを流すのが遅れた。
2. 一週間前から僕自身がヒッキーモードに突入した。
3. 募集の時点で全然集まらなかったのをそのまま野放しにしてしまった。
4. 百萬石編集開始の前に部室に tex 環境をそろえていなかった。

あたりにあると思われます。
一週間前からヒッキーモードに突入した理由のいくつかは、輸血経験者は献血できないとか言われて献血を断られたり[1]プロバイダの契約が切れて自宅がネットワークアンリーチャブルと化したり、バイト先の飲み会で、ひとり何故か簡単に潰れてサレジオ修道院の門の前で夜を明かしたりしたためだとおもわれます。

そろそろ疲れました。
おやすみなさい。

[1]実際は輸血経験はないことが 3 日後判明。だました看護婦ムカツク