2000年度

Lime

京都工芸繊維大学コンピュータ部

暗号って何ょ?

M1 服部 保

平成 12 年 11 月 17 日

1　はじめに

暗号ってなんじゃろなと思う人は多いかもしれません．広辞苑（第二版増補版）によると，暗号とは

> 通信の秘密を守るため、当事者間にのみ了解されるように決めた特殊な記号。外交・軍事・警察・商業などに使用。

となっています.

要するに，他の人が盗聴したり盗み見しても,「中身はさっぱわけわかめ」にするものです．例えば, “Khoor, zruog.” という文字列があったとき，どういう意味かわかりますか？分かりませんね．実は，これは，“Hello, world.” という文章を暗号化したものです．では，どうやって暗号化したか分かりますか？

これは，各アルファベットを 3 文字後ろにずらすことで暗号化[1] を行っています．“x, y, z” は 3 文字ずらせないので，それぞれ “a, b, c” に変わります．この暗号は，Julius Caesar が用いた「シーザー暗号」というもので，そのころから既に暗号は（特に外交・軍事目的で）使われていたわけです.

シーザー暗号の場合,「アルファベットを 3 文字後ろにずらす」という情報を知らない限り，元の文章（平文という）を得ることは出来ません．もちろん,「何文字かずらしてある」ということを知っていれば，総当たりで試せばそのうち意味のある文章が出ますから，暗号化に用いた情報は全て隠さなければいけません.

つまり,「アルファベットを後ろにずらす」という暗号化アルゴリズムと,「3 文字ずらす」という暗号化鍵を秘密にしなければならないわけです．もちろん，暗号文を平文に戻す（復号する，という）ための,「アルファベットを前にずらす」という復号アルゴリズムと「3 文字ずらす」という復号鍵も秘密にしておかないと，暗号化した意味がなくなってしまいます.

このように，古い時代の暗号は，アルゴリズムは非公開であるのが普通であり，もちろん暗号化鍵・復号鍵も秘密にしておくものでした.

ところが，いまどきの暗号は，このころの暗号とはずいぶん趣を異にしています．というのは，暗号化鍵・復号鍵は秘密にするけれども，アルゴリズムは公開するのが当然になっているからです.

ということで，いまどきの暗号についてお話してみたいと思います.

2　暗号・その分類

一口に暗号といってもいろいろあるわけで，分類しないとつかみようがないです．というわけで，ここではまず分類を行います.

歴史的な見地からの大分類として次の二つがあります.

暗号 ⎧ 古典暗号　古くから使われている暗号．アルゴリズム非公開.
　　 ⎩ 現代暗号　第二次大戦以降の暗号．アルゴリズム公開.

[1] 似た例には「2001 年・宇宙の旅」に登場する “HAL” があります.

方式としてみるときの大分類は次の二つです.

$$\text{暗号}\begin{cases}\text{秘密鍵暗号（共通鍵暗号）} & \text{暗号化鍵と復号鍵は同一．鍵は絶対秘密.}\\ \text{公開鍵暗号} & \text{暗号化鍵と復号鍵は異なる．暗号化鍵を公開.}\end{cases}$$

秘密鍵暗号は，共通鍵暗号とも呼ばれます．暗号化鍵と復号鍵が共通なためです．で，秘密鍵暗号は，さらに次のように分かれます.

$$\text{秘密鍵暗号}\begin{cases}\text{ブロック暗号} & \text{ある決まった長さのブロックごとに暗号化.}\\ \text{ストリーム暗号} & \text{平文全体に対して暗号化.}\end{cases}$$

まぁ，おおまかにこんなぐらいでよいでしょう．古典暗号の代表格と言えば．シーザー暗号です．アルゴリズム非公開という点が,「軍事用」の香りを漂わせます.

秘密鍵ブロック暗号の代表としては，DES（Data Encryption Standard），Rijndael（ラインデール，あるいはリインダールと読む）などがあります．このうち，DES はアメリカの暗号標準として，20 年来使われてきた方式で，これに代わる新規格 AES（Advanced Encryption Standard）として採用されたのが Rijndael です．他にも，三菱電機が開発した MISTY や NTT が開発した E2 などがあります.

秘密鍵ストリーム暗号の代表としては，第二次大戦中ドイツが用いた “Enigma”（エニグマ）や，日本軍が用いた “紫” 暗号があります．いずれも，軍によって作戦指揮の伝達などに用いられていたものです．秘密鍵ストリーム暗号は，ある部分を暗号化した結果を次の部分の暗号化に用いたりすることによって．非常に強力な暗号を作ることが出来るのですが，極めて長い繰返し周期を持つ擬似乱数を用いなければならないので，非常に扱いにくいために,，研究が続けられているにも関わらず実際に使われることはほとんどありません.

公開鍵暗号の代表としては，RSA（Rivest-Shamir-Adleman）暗号，ナップザック型暗号などがあります．これらについては，細かいことはあとで述べます.

と，ここで疑問に思うことはないですか？秘密鍵暗号で,「鍵を秘密にする」とわざわざ書いていますが,これは普通当たり前ですね.「うちの鍵はこんな形だよぉ」などと皆に見せびらかす人はいないですね.「暗号化鍵と復号鍵が同一」と言うのも当たり前です．形が違う鍵で開いてしまったら，鍵の意味がないですね.

ところが，公開鍵暗号は「暗号化鍵と復号鍵は異なる」上に,「暗号化鍵を公開」するというのです．どういうことでしょう?「こんなん暗号にならへんのちゃうん？」と思うでしょう？でもなるんですな，これが．これもあとで詳しく述べましょう.

このあとでは，古典暗号・ストリーム暗号に関しては，もう無視します.

3　まず攻撃から

秘密鍵暗号・公開鍵暗号について説明する前に．まず攻撃に関して述べておきます．まず，結論から言いますと,

世の中の暗号は全て解読可能である!

のです.

「な〜んやそれ．い〜みな〜いじゃ〜〜ん」と思ったでしょう？ところが，この結論には,「そんなん絶対ありえねー」仮定がありまして，それは何かと申しますと,

攻撃者は無限の計算能力を持つか,
または解読に要する時間は全く考慮しない

というものです．何を言ってるかというと，スパコンなぞゴミみたいな計算能力を持っているか，さもなくば

$$\text{解読に } 10^{100} \text{ 年かかってもいい}$$

ということで，こんなん全然現実的じゃないです．

現代暗号では，基本的に「現実的な時間では解読できないものは事実上安全である」と言うスタンスで安全性を考えています（計算量的安全性といいます）．

例えば，ライバル会社の新製品の設計図が暗号化されたものが手に入ったとしましょう．で，その発表日は半年後だとします．このとき，その暗号を解くためにどんなに高速に計算できる手法を用いても，半年以上の時間がかかってしまうのならば，暗号が解けても全く意味がありません．だって，発表日までに解読して製品を作って先に発表しないと意味がないですから．

まぁ，これは極端な例ですけど，実際に提案されてる暗号というのは現在考えうる全ての計算手段を考えても，数百万年は最低かかるので安全だ，みたいなところがあります．

現代暗号はアルゴリズムを公開していますから，復号鍵を総当たりして試していれば，そのうちうまく復号できる鍵に当たることは目に見えています．これが 16 ビットの鍵程度なら一瞬で総当たり完了ですが，64 ビットともなると難儀です．1 回復号するのに 1μ 秒かかるとして，総当たりには 2^{64} 回だけ復号を試さなければいけませんから，$2^{64}\mu$ 秒，すなわち約 58 万 5000 年かかると言う計算になります．まぁ半分ぐらいのところで当たるとしても，約 29 万 2500 年はかかるので，まず現実的には無理だと言うことが分かってもらえるでしょう．

ただ，計算量的安全性に基づく暗号は，提案されてから時間が経つに従って安全性が下がっていく傾向にあります．というのは，計算機の性能はどんどん上がっていきますから，去年の「計算能力の限界」で計算して 100 万年かかったものが，今年の「計算能力の限界」で計算すると 50 万年で済むようになったりするわけです．計算能力の限界が毎年 2 倍になるとしたら，100 万年かかったものが 1 年で済むようになるまで 20 年程度ですから，20 年経った時点で安全性は 100 万分の 1 に下がることになるわけです．つまり，暗号の寿命みたいなもんがあるわけですね．

で，攻撃と一口に言っても．いろいろあります．秘密鍵暗号に対しては，復号鍵を総当たりで見つけ出す攻撃しかありませんが，公開鍵暗号に関してはその暗号が安全性の根拠を何に置くかによって，攻撃者が付け入る隙となるポイントがそれぞれ違うので，暗号方式それぞれについて特有な攻撃が考えられています．

さてさて，最近では情報量的安全性（鍵の候補が無限にあれば，総当たりは絶対不可能）を持つ暗号を作ろうという研究もありますが，そう問屋が卸すわけがないので無理だと思うんですけどね……．これをやったら Turing 賞とか取れるんでしょうねぇ……．あ，ノーベル賞は無理ですよ，数学賞ないから．

4　秘密鍵暗号

世間一般に暗号と言って，一番暗号らしいのが秘密鍵暗号でしょう．家の鍵と一緒で，同じ鍵を持っている人しか解けませんから，実生活の感覚に一番近いでしょう．鍵の形さえ他人に教えなければ安全なのですから．

秘密鍵暗号の特徴は，

- 高速
- ハードウェア化が容易
- n 人の相手と暗号通信をするには，鍵を n 個持つ必要がある

- 安全性が証明可能

といったところでしょうか．秘密鍵暗号の基本構造は,「EXOR やビット列の入れ換えなどの 可逆 な操作を複雑に組み合わせることで暗号化する」といった感じです．この可逆と言うのがポイントです．というか，可逆じゃなかったら，ちゃんと復号できませんがね．

と，それはさておき，ビット列の入れ換えや EXOR などは非常に簡単な操作ですし，ハードウェア化が容易なのも自明ですから，特徴のうち上 2 つは分かりますね．3 つ目も当然ですが，これが欠点ともなっています．A さんと B さんに対して同じ鍵を使うと，A さんに送った暗号文を B さんが復号できてしまったりして困りますから，それぞれ別の鍵にしなければいけません．相手が 2, 3 人なら鍵の管理は簡単ですが，100 人とかになるとちょっと面倒です．というわけで,「鍵管理が大変」という欠点を持ちます．

5 つ目は，とんでもなくややこしい話なのではしょりますが，アウトラインだけ．さっき，秘密鍵暗号に対しては「復号鍵を総当たりで見つけ出す攻撃しかない」といいましたが，実は総当たりする範囲を限定する手法（これも攻撃手法になる）がありまして，それが「差分攻撃法」と「線形攻撃法」の 2 つです．どちらも，都合のいい暗号文–平文のペアを寄せ集めて，鍵の情報を得ようとか．そういう攻撃法です．

で，ここでいう証明可能というのは,「差分攻撃法や線形攻撃法を適用しても，全部の鍵を総当たりで試すよりも簡単に求めることはできないよ」と証明できるということです．どんなに都合のいい暗号文-平文のペアを持ってきても，それから導き出される「鍵のあるビットが'0' か'1' かの確率」が 1/2 なら，特定は不可能ですからね．

で，秘密鍵暗号の代表格であるところの DES の話が続きます．

4.1 DES

DES は 1977 年にアメリカ合衆国データ暗号化標準に制定された方式です．

アメリカの標準は世界の標準

みたいなところがありますし，なんせ当時のアメリカは暗号では世界をリードしてましたから，世界中に広まって使われています．

特に，金融分野での情報の送受には広く使われているそうです．アメリカでは，FRB（Federal Reserve Banks，アメリカ連邦準備銀行）と各金融機関の間の通信に DES が使われているとのことです．

歴史的な背景を見ておきますと，1973 年に，DES の制定に当たって公募が行われ，その際 IBM が提案した Lucifer（ルシファー）をベースとして，NBS（National Bureau of Standard）[2] などによる審査や改良を経て，1977 年に DES として完成されました．

「公募開始から制定まで 4 年かかった」ということから,「政府機関が勝手に解読できるような仕組みが秘密裏に組み込まれたのではないか？」というまことしやかな噂が流れたりしたのですが，23 年経つ今もその仕組みは見つかっていないので，多分ないのでしょう．あったらえらいことなんですけどねぇ．

それはさておき，Solaris なんかを使っている人には DES はおなじみではないかと思います．というのは，パスワードの暗号化に DES が用いられているので．まぁ，サーバが Solaris でクライアントが FreeBSD なところに NIS なんか入れたりすると FreeBSD はパスワードの暗号化に MD5 を用いているのでパスワードファイルが共有できねぇと言う憂き目にあって，泣きそうになりながら DES の配布パッケージを ftp かなんかで取ってきて入れようとすると DES はアメリカが輸出規制をかけてるせいで限られたサーバにしか置いてないのを忘れてて他のところにいってしまって時間を無駄にしてどこかから「あんたバカぁ？」なんて声が聞こえる気がしたりしなかったりしてさらに泣きそうになるという素敵なこともあったりしたけど，今では輸出規制はゆるくなってて何も気にせず使えるのでラッキーって感じ（スタパ風）．

[2] 米国内における科学技術全般の標準規格を策定する政府機関．現在は，NIST（National Institute of Standards and Technology）と改称されている．

で，DES は 56 ビット（ただし，7 ビットごとにパリティを付加するので利用の際は 64 ビット）の鍵を用いて，64 ビットごとの平文ブロックを暗号化する方式です．暗号化の方式自体には触れないので，興味を持った方はジュンク堂などに言って暗号関係の本でも買ってください．

DES は 20 年来使われ続けていますが，これが破られたことによる被害というのはないみたいです．とはいうものの，実際のところは研究者により専用ハードウェアを使ったり，特殊な高速化技法を使ったりという方法で鍵の総当たりをされて数週間とか数日とかいう単位で破られるようになっています．今年の 9 月まで，郵政省所轄の通信・放送機構（TAO）に「情報通信セキュリティ技術研究開発プロジェクト」というのがありまして，わずか 3.7 日で DES の鍵総当たりによる解読を完了したという成果が出ています．これには，ちょっとゴツい並列計算機 AP3000 と DEC ALPHA 21164 500MHz を使用して，改良型線形攻撃法により鍵候補を 2^40 個程度まで絞り込んで行ったそうです（ちなみに，鍵の総当たり自体には 4 時間しかかかっていない）．

まぁ，普通の人がこんな計算機環境を持ってるわけがないので，誰にでも DES が破れるというわけではないですが，国家レベルの予算があれば，それこそ数時間単位で DES を解読できる計算機環境を作り上げるぐらいのことはわけもないことなので，おそろしいもんです．

5　公開鍵暗号

公開鍵暗号は，1976 年に W. Diffie と M. E. Hellman がその概念を提案したもので，分野としてごく新しいものです．古来から，暗号化の鍵は絶対秘密にしておくものだ，ということが

究極不変の真理

のように思われていたのですが，公開鍵暗号では

暗号化鍵を見せびらかしてもよい

という，それまでの常識を根底から覆すようなことを言っています．

暗号化鍵と復号鍵が別で，暗号化鍵を見せびらかしてもよくて，しかもちゃんと暗号として成り立つなんて,「そんな仕組みが実現できるのか？」と思うでしょう？でも，できるんですな，これが．

平文を m，暗号文を c，暗号化関数を $E()$，復号関数を $D()$，暗号化鍵を k_e，復号鍵を k_d とします．

公開鍵暗号では，$E()$ に一方向性関数というものを用います．一方向性関数とは，それ自身を計算することは簡単だが，逆関数を求めることは極めて困難であるような関数のことを言います．

公開鍵暗号で用いる場合は，$c = E(m, k_e)$ に対して逆関数 $E^{-1}(c, k_e)$ を求めることが困難であればよいわけです．そして，復号関数は $m = D(c, k_d)$ となればよいわけです．

要するに，

暗号化関数の逆関数が求まらないから，
復号鍵を知らない限り解けない

ということになっています．

どうやったら，そんな問題を作れるのかというのは，これから述べていきます．

で，公開鍵暗号の特徴ですが，

- n 人の相手と暗号通信をするときでも，相手の鍵を保存する必要がない．
- 低速

- 原理が複雑
- ハードウェア化が困難
- 安全性の証明が困難

1 つ目は公開鍵暗号ならではの特徴ですが，相手の暗号化鍵は公開されているので，必要になったらそのときに取りに行けば事足ります．しかも，その鍵を秘密にしておく必要は一切ありません．秘密にしておかなければならないのは自分の復号鍵だけ，というわけで,「鍵管理が簡単」という長所になります．

2 つ目と 3 つ目は関係しますが，暗号化鍵を公開してもいいようにするために，いろいろと数学的な工夫が凝らされているので，複雑な処理を必要とします．処理が複雑ということは時間がかかるということですから，低速になるというわけです．また．処理が複雑になると，ハードウェア化も難しくなってしまいます．

公開鍵暗号は，安全性の証明が極めて困難です．というのは，復号鍵の総当たりだけでなく，暗号化鍵から復号鍵を求めるような攻撃が可能か否かや，暗号文と暗号化鍵から平文が求められたりはしないかなど，検討しなければならない課題が多い上，別の学問分野で用いられているアルゴリズムを用いると効率よく解読されてしまう例があったりして，どこから証明してよいかが分からないのが現状です．もっとも早くから提案されている RSA 暗号でさえ，その解読の困難さが,「非常に大きな 2 つの素数の積の素因数分解を行うことと同等程度らしい」ということが知られているだけで，安全性の証明はおろか「解読の困難さ=素因数分解の困難さ」であることすら証明されていません．いくつか，証明しようとした例はあるものの，不備があってどれも成功していません．

このように，欠点はあるものの，不特定多数の相手との通信を，鍵の管理を考えることなしに行えるというメリットがあるため，いろいろな研究がなされています．

代表格の RSA 暗号についてちょっと詳しく触っておきます．公開鍵暗号は，イメージつかみにくいと思いますので．

5.1 RSA 暗号

RSA 暗号は，1978 年に，R. L. Rivest，A. Shamir，L. Adleman の 3 人により提案された公開鍵暗号で，公開鍵暗号の代名詞とも言える存在です．

これは，非常に大きな合成数の素因数分解問題の一方向性を利用しています．ここで，合成数とは，素数を掛け合わせてできる数です．素数については，中学校時代の知識を動員してください.「1 とその数自身でしか割り切れない数」でしたね．

つまり，非常に大きな素数 p, q を知っていて，それを掛け合わせて合成数 N を得ることは非常に簡単ですが，その逆，すなわち非常に大きな合成数 N が与えられているときに，その素因数 p, q を求めるのは非常に困難である，ということを利用しています．

準備

1. まず，非常に大きな 2 つの素数 p, q を選ぶ．
2. その素数の積 $N(= pq)$ を求める．
3. N のオイラー関数 $\varphi(N) = (p-1)(q-1)$ を求める．
4. $\begin{cases} \gcd(d, \varphi(N)) = 1 \\ \max(p, q) < d < \varphi(N) \end{cases}$
   となるような d を求める．ここで，$\gcd(x, y)$ は x と y の最大公約数である．

5. $ed \equiv 1 \pmod{\varphi(N)}$ となる e を求める．ここで，$x \equiv y \pmod{z}$ は，
   x を z で割った余りが y (を z で割った余り) と等しいことを意味する．
6. 暗号化鍵として N, e を公開する．d は復号鍵として秘密に保持する．

暗号化

1. 平文 m を N 以下の数に変換し，それを M とする．
2. 公開されている N と e を用いて，暗号文 C を

$$C \equiv M^e \pmod{N}$$

   として求め，相手に送信する．

復号

1. N と復号鍵 d を用いて，M を

$$M \equiv C^d \equiv (M^e)^d \equiv M^1 \pmod{N}$$

   として求める．
2. M を平文 m に変換する．

こう長々書かれると訳分からないでしょうから，分かりやすさのためにちょいと例を考えましょう．一瞬で破れる例ですが．

1. $p = 11, q = 7$ とします．
2. $N = pq = 77$ です．
3. $\varphi(N) = (11-1)(7-1) = 60$ です．
4. $11 < d < 60$ で $\gcd(d, 60)$ を満たすものから選んで $d = 37$ とします．
5. $37e \equiv 1 \pmod{60}$ となる e は 13 ですから $e = 13$ になります．
6. 暗号化鍵として $N = 77$, $e = 13$ を公開します．

送信者側は……

7. 平文 m を変換した結果 $M = 70$ を得たとします．
8. $70^{13} \equiv 42 \pmod{77}$ より，$C = 42$ を送信します．

受信者側は……

9. $42^{37} \equiv 70 \pmod{77}$ より，$M = 70$ を復号します．

とまぁ，こんな感じです．
RSA 暗号は，

N の素因数分解が困難

↓

p, q が求まらない

↓

$\varphi(N)$ が求まらない

↓

e から $ed \equiv 1 \pmod{\varphi(N)}$ となる d が求まらない

↓

安全

という論法で安全性を保障しています．今のところ，p, q は 512 ビット程度にとれば実用上安全であるとされています．ちなみに，512 ビットの数というと，10 進数で 155 桁ぐらいある数になります．

RSA 暗号のアルゴリズムについては，特許が取得されていましたが，2000 年 9 月 20 日に特許の有効期限が切れたため，自由に使えます．

メールの暗号化ソフトとして有名な PGP（Pretty Good Privacy）は，RSA 暗号が用いられていたせいで特許問題などの関係で大もめしました．作者の Philip R. Zimmerman がネットワーク上でフリーで流したため，RSA 暗号の特許権を持つ RSA 社との間で紛争になりました．

ネットワーク上でフリーで流されてしまったものを，後からやめさせることは出来ないので，RSA 社側が譲歩し，暗号化・復号部分に RSA 社製の暗号化ライブラリ（RSAREF）を用いることを条件に RSA 暗号の使用を認めました．この RSAREF ですが，わざわざ遅くなるように作ってありまして，顧客は無料で遅い PGP か有料だが速い RSA 社製暗号化ソフトかを選ぶということになりました．まぁ，普通は遅くてもフリーのを使いますがねぇ．

問題はそれだけではなく，今度はアメリカの暗号輸出規制が PGP の前に立ちはだかりました．しかし，これはアクロバティックな方法によりあっさり回避されてしまいました．というのは,「暗号化ソフトウェアは輸出規制に抵触するが，アルゴリズムを記した出版物は，輸出規制の範囲外である」という暗号輸出規制の盲点をついて，ソースコードを出版して国外に輸出し，そのソースコードを暗号輸出規制のない国でコンパイル・配布するという手段を用いたのです．その結果，世界中で PGP が法的な問題にさえぎられることなく使えるようになったというわけです．

公開鍵暗号には，他にもいろいろあるんですが，話すと 5m 以上になるので省略させてもらいます．私が研究してるのは，積和型暗号といわれる部類の公開鍵暗号なんですけど，なにせバリエーションが多すぎて…….

6　暗号の応用例

暗号は，何も他人に見せたくない情報を隠すためにだけ使うものではありません.「このメールは確かに自分が書いたものである」という証明をしたりすることもできます．というより，むしろ現在ではそちらの用途が広く使われていくと考えられています.

「このメールは確かに自分が書いたものである」と証明をする技術，それが電子署名です.

6.1　電子署名

電子署名は，あるメッセージを自分が書いたことを証明する必要がある際に用いる技術です．普通の書類なら，印鑑を押したりサインをしたりして自分が書いたことを証明しますが，ディジタル文書に対して印鑑を押したりサインしたりは出来ません.

ところが，ネットワーク上でも自分が書いたことを証明しなければならないことがあります．そこで，実世界での印鑑やサインに変わるものとして，電子的な署名，すなわち電子署名が必要になるわけです．

電子署名は公開鍵暗号を応用して，自分が書いたことの証明を与えようというものです．なぜ，公開鍵暗号が出てくるかというと，公開鍵暗号では公開鍵（暗号化鍵）と秘密鍵（復号鍵）がありますから，自分が書いたことを他の人に確認してもらうにはちょうどよいからです．つまり，A さんが自分の秘密鍵でメッセージを暗号化しておいて，他の人が公開鍵で復号できれば,「この公開鍵でちゃんと復号できるような暗号文を作れる秘密鍵を持っているのは A さんだけなので，確かに A さんが書いたものだ」ということが分かるという仕組みです．

実際にはこんな感じで使われます．A さんが B さんに電子メールを送るときに，署名をつける場合を考えてみます．

1. A さんは，電子メールを書く．
2. A さんは，その電子メールの内容（または，そのダイジェスト）を自分の秘密鍵で暗号化したものを署名として電子メールに付加し， B さんに送信する．
3. B さんは，受け取った電子メールに付加された署名を A さんの公開鍵で復号する．
4. 署名を復号した結果と，受け取った電子メールの本文（または，そのダイジェスト）と比較し，一致すれば確かに A さんが書いたものだと分かる．

とまぁ，こんな具合で使うわけです．

この方法は，RSA 暗号と非常によくマッチします．というのは，RSA 暗号では暗号化アルゴリズムと復号アルゴリズムが同じで，暗号化鍵の e と d は互いに入れ替えてもまったく問題ないからです．

7　最近のトピック

ちょっと趣向を変えて，最近のトピックなど．というか，RSA 暗号の特許切れはさっき書いちゃったので使えませんなぁ．ということもあって，AES 話と TAO 話でもしますか．

7.1　AES

最近の計算機の性能向上のため，DES の安全性の保証はかなり低下してしまいました．

「あなたの秘密は 3 日しか守れません」

と言われて，DES を使う人がいるわけがありません．DES を暗号化標準としているアメリカとしても，普通の人にまで破られるようになってしまう前に，DES より強力な暗号に乗り換えたいところです．そうでないと，国防秘密やらなんやらがダダ漏れになってしまいますから．

というわけで，1998 年に予定されていた DES の再評価を前に，新しい暗号化標準である AES（Advanced Encryption Standard）を選定することが決定されました．

どういう流れで選定作業が行われたかと言うと…….

1997 年 1 月 2 日
NIST により AES 公募の方針発表．要求条件案と選定基準案の発表．

1997 年 9 月
正式な要求条件と選定基準の公開，公募開始．

1998 年 6 月 15 日
公募締め切り．15 件の応募がある．

1998 年 8 月 20〜22 日
第 1 回選考会．各暗号方式のプレゼンテーション．これをもとに安全性などの評価が始まる．

1999 年 3 月 22,23 日
第 2 回選考会．第 1 回のプレゼンを元にした安全性などの評価結果発表．

1999 年 8 月 9 日
5 つの最終候補発表．

2000 年 4 月 13,14 日
第 3 回選考会．最終候補についてのより詳細な評価結果報告．

2000 年 10 月 2 日
AES として Rijndael が正式決定される．

という感じです．
で，NIST による AES 候補に対する要求条件の正式版は，次の通りでした．

- 共通鍵暗号であること
- ブロック暗号であること
- 鍵長は 128 bit，192 bit と 256 bit が利用可能であること．
- ブロック長は 128 bit が利用可能であること．

要するに，DES の 2 倍の鍵長・ブロック長を持つ秘密鍵ブロック暗号を作れってことですな．平文を長くしたのは，解読に都合のいい暗号文-平文対の組み合わせもとんでもない数にしちまえという発想です．鍵の長さが 2 倍になるだけでも，鍵の候補は 2 乗倍になりますから，そりゃあなた,「解読なんてもう無理ですわ〜」というとこまで持っていけますわな．
で，選定基準は優先度の高い順に次の通りでした．

1. 安全性
   他の候補との比較や，安全性の数学的根拠について検討
2. コスト
   ライセンス要件，処理速度，メモリ容量
3. その他アルゴリズム的特徴
   使用範囲の広さ，処理の単純さ等

まぁ，いくら鍵の長さを長くしても，これまでに発表されている攻撃手法が効率よく効きまくるようではまいっちんぐなわけでして，それらに対しては強くないといかん，と．で，他の候補と比べて強いものを選ぼうと言うのも当然ですわな．わざわざ弱いものを選ぶのはアホですから．
で，コスト面では，ライセンス要件と言うのが目を引いてます．これは，もし，応募された暗号のアルゴリズムなどにどこかの特許が含まれていたとき，その暗号が AES として採用された場合にはその特許権を行使しないという誓約を取るべし，というものでした．NIST としては，AES は完全にライセンスフリーで無償使用できるものにしようと考えたようです．処理速度は時間的コスト，メモリ容量は空間的・金銭的コストですね．

「その他アルゴリズム的特徴」というのは漠然としていますが，まぁ応用範囲が広いだとか，アルゴリズム的に面白みがあるだとか，処理が単純明快だとか．そういうことです．

で，こういった条件を満たすものとして，1998 年 6 月 15 日の締切までに，15 件の応募がありました．

- CAST-256（Entrust Technologies, Inc.）
- CRYPTON（Future Systems, Inc.）
- DEAL（Richard Outerbridge, Lars Knudsen）
- DFC（CNRS - Centre National pour la Recherche Scientifique - Ecole Normale Superieure）
- E2（NTT - Nippon Telegraph and Telephone Corporation）
- FROG（TecApro Internacional S.A.）
- HPC（Rich Schroeppel）
- LOKI97（Lawrie Brown, Josef Pieprzyk, Jennifer Seberry）
- MAGENTA（Deutsche Telekom AG）
- MARS（IBM）
- RC6（RSA Laboratories）
- Rijndael（Joan Daemen, Vincent Rijmen）
- SAFER+（Cylink Corporation）
- SERPENT（Ross Anderson, Eli Biham, Lars Knudsen）
- TWOFISH（Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson）

で，1998 年 8 月 20〜22 日の第 1 回選考会で上の各暗号の説明が行われました．まぁ，物事にはドラマが付き物でして，AES 選考に関しても例外ではなかったようです．FROG，MAGENTA と LOKI97 はこのときに落選したのですが，LOKI97 は，なんと公募締切日に弱点が指摘されてしまい，落選．FROG も選考会前に同じく AES 候補の “TWOFISH” のグループにより弱点が指摘されて落選．MAGENTA は選考会の席上で猛烈なディスカッションが行われた結果破れることが分かり，翌日には解読論文が会場で配布されたとのこと．ちなみに，この解読論文には RSA 暗号の発明者の一人，Adi Shamir が名を連ねており，彼によると論文として内容をまとめるのには 15 分程度だったそうな．

1999 年 3 月 22,23 日の第 2 回選考会では，第 1 回で行われた各暗号の解説を元に，安全性や処理速度についての評価結果が発表されました．この評価結果を元に比較検討が行われ，1999 年 8 月 9 日に次の 5 つが最終候補とされました．

- MARS
- RC6
- Rijndael
- SERPENT

- TWOFISH

2000 年 4 月 13,14 日に行われた第 3 回選考会では，この 5 つについて安全性などについてのより突っ込んだ評価と議論が行われまして，最終的に 2000 年 10 月 2 日に Rijndael に決定の運びとなったわけです.

で，AES に決定した Rijndael ですが，ソフトウェア実装における高速性とメモリ使用量の少なさに定評があります．実は，第 2 回の時点で Rijndael でほぼ決定だろうと言われていたようですが，最終候補は 5 つでした．これは，IBM の MARS を残したかったからだろうと言う意見も一部にはありまして，というのも他国製の暗号を自国の標準にしたくないのではないか，という至極当然な推測が元になっています．さすがに Rijndael と MARS の 2 つを最終候補にすると,「なんでやねん」という突っ込みがきそうだから，5 つにしてぼかしたらしいというのですが……．どうなんだろ？

で，興味がある方は，こちら．`http://csrc.nist.gov/encryption/aes/`．当然のことながら英語ページですけど.

7.2 TAO

TAO っていうのは，郵政省の認可法人であるところの通信放送機構[3] （http://www.shiba.tao.go.jp）の略称です.

平成 7 年度から 12 年度までの 5ヵ年計画で,「情報通信セキュリティ技術研究開発プロジェクト」というプロジェクトが動いてまして，このプロジェクトではいろいろと国際的にも重要な成果が上がっていました.

このため，5ヵ年計画だったのを棚上げして，このまま継続しようという声が，国内の暗号研究者の間から出たのですが，なにせ国家予算に基づくプロジェクト，お役所の「5ヵ年計画ということでやっていたので」という，将来も何も考えてないコメントで一蹴されてしまいました.

猫も杓子も IT 革命だと大騒ぎになりだす前だった（森総理が IT 革命だとか言い出すより前，というか就任よりもずっと前です）ので，結局プロジェクトは継続されることなく終了してしまいました.

大きな成果を挙げた情報セキュリティプロジェクトを「5ヵ年計画だったから」などという理由で終了させておいて,「とにかく IT 革命だ！」って騒いでる政府っていったい何？IT の中には，情報セキュリティ技術も含まれていることを忘れている（というか分かってない？）んですかねぇ？IT 先進国となることを目指すなら，情報セキュリティ先進国であることも目指さなきゃいけないんですけどね.

というか，私としては電子選挙を実現して欲しいんですけど．実現したら，雨が降ったからって投票率が落ちることもないし，開票・集計作業は一瞬で終わるし．ついでに，選挙活動もネット上でやれば，選挙資金もかからなくなっていい感じなんですけどねぇ．やっぱり，IT 先進国を標榜するなら，これぐらいしないと．もちろん，選挙区での演説はそれなりにちゃんとやってもらわないと困りますが．少なくとも，電子選挙にすれば，アメリカ大統領選挙みたいなことにはならないと思うんですけど，どうでしょう？

[3] Telecommunications Advancement Organization of Japan

日本語.comってステキ…

岐津三泰

平成 12 年 11 月 20 日

概 要

皆さんが普段インターネットの Web ページにアクセスする時に使うアドレスが日本語だったらいいな、と思ったことはありませんか？例えば郵政省の Web ページのアドレスが http://www.mpt.go.jp/よりも http://郵政省.jp/の方がより親しみ安くて覚えやすいですよね。もともとこの名前 (ドメイン名) はアルファベットと数字とハイフンしか使えない約束になっていますが、最近、多言語に対応しようという動きがあります。去る 11/9 には.com/.net/.org について登録が開始されました。今回はその技術のさわりを紹介します。

1 多言語ドメイン名の仕組み

ドメイン名を多言語化するには DNS サーバで使うコードセットを規定する必要があります。大きく分けて、従来ドメイン名に使える文字だけを使う方法と、従来のドメイン名としては正しくない文字を使う方法があります。現在は前者の方が支持されているようです。

1.1 エンコーディング相互変換

DNS で使う文字セットを従来のままにするにしろ拡張するにしろ、処理系によって違う文字セット (ローカルエンコーディング) を使用する為、アドレスを索く際にどこかでエンコーディングの変換を行なわなければいけません。これには次の 3 種類が考えられます。

1. クライアントでエンコーディングの変換をする。
2. DNS サーバでエンコーディングの変換をする。
3. クライアントと DNS サーバの間に DNS プロキシサーバを置き、それがエンコーディングの変換をする。

実際、2 の方法は DNS に大きな負荷を与えることになる為に好ましくありません。後述する mDNkit でも 2 を除いた方法が提供されています。

1.2 RACE 変換

多言語文字を従来のドメイン名に使える文字だけに変換する方法 (ACE; Ascii Compatible Encoding) はいろいろ提案されていますが、現在有力なのは RACE(Row-based ACE) です。RACE は入力として Unicode のエンコーディングの一つである UTF-16 を想定してい

ます。UTF-16 は Unicode の最初の 65536 文字を 2 バイトで、残りは 4 バイトで表現されます。また、変換する前に正規化という作業を行なわなければいけません。ここでいう正規化とは、同じ表現を数種類持つ文字を 1 つに統一することです。例えばアルファベットの大文字や全角文字を半角小文字にしたり、半角カナを全角にしたりといったことです。つまり、実際の変換作業は、ローカルエンコーディングから UTF-8 に変換 → 正規化 → UTF-16 に変換 → RACE 変換という順序で行なわれます。

RACE へのエンコードは次のようなステップで行なわれます。

1.2.1 圧縮

まず、UTF-16 の入力を次のような手順で圧縮します。入力は 2 オクテットごとのセットとみなします。

1. 入力すべての上位オクテット (U1 とする) が同じなら 4 へ行く。
2. 入力すべての上位オクテットが 0 とそれ以外の一種類 (U1 とする) なら 4 へ行く。
3. 0xD8 を出力し、入力すべてを出力して終了。
4. U1 を出力する。
5. もし入力の終端であれば終了。そうでなければ次のオクテットを読み込み (U2 とする)、その次のオクテットも読み込む (N1 とする)。
6. U2 が U1 と等しく、N1 が 0xFF でなければ N1 を出力し、5 へ行く。
7. U2 が U1 と等しく、N1 が 0xFF なら 0xFF、0x99 を出力し、5 へ行く。
8. 0xFF、N1 を出力し、5 へ行く。

これを C で書くと以下のリストのようになります。

```
#define NOCOMPRESS 1
#define COMPRESS 0

int
compress(len, inbuf, outbuf)
     unsigned len;           /* 入力バイト数 / 2 */
     unsigned short *inbuf;  /* UTF-16 の入力 */
     unsigned char *outbuf;  /* 出力 */
{

  unsigned short u, upper = 0;
  unsigned char u1, u2, n2;
  int i, mode = COMPRESS, outlen = 0;

  for(i = 0; i < len; i++)      /* Step 1,2 */
    {

      u = inbuf[i];
```

```
    if(u == 0)
      continue;
    if(upper == 0)
      {
        upper = u;
        continue;
      }
    if(u != upper)
      {
        mode = NOCOMPRESS;
        break;
      }

  }

if(mode == NOCOMPRESS)
  {

    outbuf[outlen++] = 0xd8;                              /* step 3 */

    for(i = 0; i < len; i++)
      {

        u2 = (unsigned char)((inbuf[i] >> 8) & 0xff);
        n1 = (unsigned char)(inbuf[i] & 0xff);
        outbuf[outlen++] = u2;
        outbuf[outlen++] = n1;

      }

    return outlen;

  }
else /* mode == COMPRESS */
  {

    u1 = (unsigned char)((upper >> 8) & 0xff);
    outbuf[outlen++] = u1;                                /* step 4 */

    for(i = 0; i < len; i++)
```

```
        {

          u2 = (unsigned char)((inbuf[i] >> 8) & 0xff);  /* step 5 */
          n1 = (unsigned char)(inbuf[i] & 0xff);

          if(u2 == u1)
            {

              if(n1 != 0xff)
                {
                  outbuf[outlen++] = n1;                /* step 6 */
                }
              else /* n1 == 0xff */
                {
                  outbuf[outlen++] = 0xff;              /* step 7 */
                  outbuf[outlen++] = 0x99;
                }

            }
          else /* u2 != u1 */
            {

              outbuf[outlen++] = 0xff;                  /* step 8 */
              outbuf[outlen++] = n1;

            }

        }

      return outlen;

    }

}      /* End */
```

圧縮の例を示すと、

- 上位オクテットが全て同じ場合
  <012e><0110><014a> → 0x012e104a
- 上位オクテットが 0x00 とそれ以外の 1 種類の場合
  <012e><00d0><014a> → 0x012effd04a

- 上位オクテットが全て同じで、下位オクテットに 0xff が含まれる場合
  `<1290><12ff><120c>` → `0x1290ff990c`
- 上位オクテットが 0x00 を除いて 2 種類以上の場合
  `<012e><00d0><24c3>` → `0xd8012e00d024c3`

1.2.2 長さ検査

圧縮後の文字列の長さが 36 オクテットを越えていないかチェックします。37 オクテット以上はエラーとなり、変換に失敗します。

1.2.3 Base32 エンコード

圧縮した文字列を次の手順で Base32 エンコードします。

1. 入力から 5 ビット読み込む。但し、入力の残りが 5 ビットに満たない場合は足りない分を 0 でパディングする。
2. 表 1 に従って、一文字出力する。
3. もし入力の終端なら終了。そうでなければ 1 に戻る。

表 1: Base32 変換表

ビット	文字	ビット	文字	ビット	文字
00000	a	01011	l	10110	w
00001	b	01100	m	10111	x
00010	c	01101	n	11000	y
00011	d	01110	o	11001	z
00100	e	01111	p	11010	2
00101	f	10000	q	11011	3
00110	g	10001	r	11100	4
00111	h	10010	s	11101	5
01000	i	10011	t	11110	6
01001	j	10100	u	11111	7
01010	k	10101	v		

Base32 エンコードの例を示すと、

- 入力のビット数が 5 で割り切れる場合
  `0x1290ff990c` →
  `00010010 10010000 11111111 10011001 00001100` →
  `00010 01010 01000 01111 11111 00110 01000 01100` → `ckip7gim`
- 入力のビット数が 5 で割り切れない場合
  `0x012e104a` →

```
00000001 00101110 00010000 01001010 →
00000 00100 10111 00001 00000 10010 10000 → aexbasq
```

1.2.4 プリフィクス

Base32 エンコードした文字列に「bq--」を前置します。
以上の流れで RACE エンコードが行なわれます。

2 mDNkit を用いた運用試験

今すぐ多言語ドメインを試す方法として JPNIC が配布している多言語ドメイン名評価キット (以下 mDNkit) を使用して少しだけ運用試験をしてみました。DNS サーバに使用した OS は FreeBSD 3.3-RELEASE、DNS プロキシサーバに使用した OS は FreeBSD 4.1-RELEASE、クライアントに使用した OS は Windows 2000 Pro(SP1)、Windows 98(1ED)、FreeBSD 3.1-RELEASE です。

2.1 mDNkit の make

まず、mDNkit[1] を make するには iconv() が必要です。FreeBSD 4.1-RELEASE の libc にもまだ含まれていないので、libiconv[2] をインストールする必要があります。次に mDNkit のアーカイブを展開し、mDNkit のトップディレクトリ (lib というディレクトリが見える所) で次のパッチを当てます。

localencoding.patch

```
*** lib/localencoding.c Wed Sep 20 11:47:31 2000
--- lib/localencoding.c Tue Nov 14 01:17:10 2000
***************
*** 138,144 ****
   }
   (void)(
  #if HAVE_SETLOCALE
!  (name = setlocale(LC_CTYPE, NULL)) ||
  #endif
   (name = getenv("LC_ALL")) ||
   (name = getenv("LC_CTYPE")) ||
--- 138,144 ----
   }
   (void)(
  #if HAVE_SETLOCALE
```

[1] http://www.nic.ad.jp/jp/research/idn/mdnkit/dist/mdnkit-1.1-src.tar.gz
[2] ftp://ftp.ilog.fr/pub/Users/haible/gnu/libiconv-1.4.tar.gz

```
!  (name = setlocale(LC_CTYPE, "")) ||
  #endif
   (name = getenv("LC_ALL")) ||
   (name = getenv("LC_CTYPE")) ||
```

このパッチは libmdn に含まれるローカルエンコーディングを自動取得するルーチンのバグを訂正するものです。make までの一連のコマンド (パッチを当てる部分も含めて) を以下に示します。

```
% fetch http://www.nic.ad.jp/jp/research/idn/mdnkit/dist/mdnkit-1.1-src.tar.gz
% tar xfz mdnkit-1.1-src.tar.gz
% cd mdnkit-1.1-src
% patch -p0 < ../localencoding.patch
% setenv CFLAGS -I/usr/local/include
% ./configure --with-iconv="-L/usr/local/lib -R/usr/local/lib -liconv"
% make
# make install
```

2.2 DNS サーバの設定

設定といっても、ZONE ファイルを新たに書くだけです。まず、日本語で ZONE ファイルを記述します (IP アドレスはもちろん架空のものです。369 なんて値はあり得ませんし。)。

```
@  IN SOA  ns.万猫.org.   root.万猫.org. (
                         2000090206
                         86400
                         3600
                         3600000
                         3600
        )
        IN      A       210.154.369.188
        IN      NS      ns.万猫.org.


ns      IN      A       210.154.369.188
IN      MX 1    ns

www             IN CNAME        ns
ヌクヌク        IN CNAME        ns
```

これを dnsconv で日本語部分を RACE に変換します。

```
@  IN SOA  ns.bq--3bhao4zl.org.   root.bq--3bhao4zl.org. (
                         2000090206
```

```
                        86400
                        3600
                        3600000
                        3600
        )
        IN      A       210.154.369.188
IN      NS      ns.bq--3bhao4zl.org.


ns      IN      A      210.154.369.188
IN      MX 1   ns

www             IN CNAME        ns
bq--gdgk7tfp    IN CNAME        ns
```

これを ZONE ファイルとして使用します。

2.3 DNS プロキシを使ってみる

DNS プロキシの設定は簡単です。/usr/local/etc/dnsproxy.conf.sample を dnsproxy.conf にコピーして編集します。重要な部分は forward と client-translation の 2 つです。forward には普段使っている DNS サーバのアドレスと必要であればポート番号を、client-translation はローカルエンコーディング名を指定します。具体的にはクライアントが Windows98 なら SHIFT-JIS、Windows2000 なら UTF-8、FreeBSD なら EUC-JP を指定しました。DNS プロキシを使う上での問題点は、今の所 OS 毎に DNS プロキシを置く必要があることと、リゾルバが 8bit クリーンでないといけないという点です。FreeBSD 標準の BIND8 は 8bit クリーンではないので、mDNkit に附属するパッチを当てることで 8bit クリーンになりますが、リゾルバはライブラリで提供されるので、アプリケーション自体をリンクし直さなくてはいけなくなります。これは大変面倒なので BIND8 に附属の nslookup を使用して万猫.org を索くことに成功しました。Windows98、Windows2000 のリゾルバは 8bit クリーンのようで、標準の ping.exe などで索けることを確認しましたが、InternetExplorer 5.5 では索くことができませんでした。

2.4 runmdn を使ってみる

runmdn は ld の Preload 機能を使って、リゾルバを多言語ドメイン対応にするものです。/usr/local/etc/mdnres.conf.sample を mdnres.conf にコピーすれば設定は終りです。あとは runmdn telnet ヌクヌク. 万猫.org とやれば、ヌクヌク. 万猫.org に telnet 出来ます。ただし、ping や rsh のような SETUID ビットが立っているコマンドは LD_PRELOAD 機能が使えない為、この方法では多言語ドメイン対応にはなりません。また、nslookup のような独自のリゾルバをもつアプリケーションでも使用できません。

3　最後に

私ははっきりいって多言語ドメイン名の必要性には疑問を感じています。実際に入力する際にいちいち変換したりするのは大変面倒だからです。でも一般の人から見れば日本語のドメイン名って使いやすいのかなぁ。ま、楽しいことは楽しいからそれでヨシとするかな。万猫.org も取ってしまったしな…。

___/____/______/_______/_________/_______/______/____/___/
Mitsuhiro KIZU -**- Kyoto Inst. of Tech. Computer Club
岐津三泰 As 天叢雲剣 @ 京都工芸繊維大学コンピュータ部
E-mail : murakumo@kitcc.org murakumo@mail2.dddd.ne.jp
murakumo@pdx.ne.jp (H"; up to 2000 charactors.)
________/______/_____/____/___/____/_____/______/________/

トラブル日記

機械システム工学科　3回生　畑山直也

○スケルトンの罠 (4月24日)

先日自作ＰＣのパーツを買いに行った。マウスをどれにしようかと思い棚を見るとスケルトンの水色のホイールマウスが並んでいた。最近はiMacに代表されるスケルトンブームである。そこで、ＰＳ/２のホイールマウスとしては一番安かったこともあり、そのスケルトンマウスを購入した。自作ＰＣは順調に組みあがり、ＯＳのインストールなど全て順調であった。

数日後の朝、その事件は起こった。突然マウスがコントロールを失ったのである。あるときには横方向しか効かなかったり、またあるときにはカーソルがまったく動かない状況に陥った。当初、マウスボール周辺のトラブルかと思われたが、その原因らしいごみや埃などは発見できなかった。その夜、なぜかマウスが復活。原因不明のマウストラブルは原因不明のまま幕を閉じるかと思われた。しかし次の日の朝、またしてもマウスが動かなくなった。たび重なるマウスの不調に苛立ちを覚えながらもマウスの上ぶたを開けて様子を見る。異常なし。

もはや買い換えるしかないのか・・・とあれこれ考えていると、重大なことに気がついた。「そうだ。事件が起こるのはいつも朝だ。」それではなぜ朝になるとマウスが動かなくなるのか？そもそもマウスはボールを動かすことによって軸を回転させ、その回転軸についているスリット付き円盤を回転させる。そしてそのスリットを発光ダイオードとフォトトランジスタで挟んだものによって検出して、マウスの変位を測定するという仕組みである。よって、明るいところではノイズが入り正常な動作ができなくなってしまうと考えられる。試しにマウス全体を手で覆いながら動かしてみると思ったとおりにまともにカーソルが動いてくれた。後はマウスの近くに光をさえぎるついたてのようなものを置けば万事解決である。

このマウスが光透過性のボディーを持っていなければこんな事件は起こらなかっただろう。そう、これはスケルトンブームが仕掛けた罠だったのだ。

後日談：その後スケルトンマウスを人に譲り、光学マウスを購入した。こいつはボールがないマウスで絨毯の上でも動いてくれるという優れものだ。掃除する必要もないので非常に楽である。価格も2000円程度とお手ごろなので今後は光学式が主流になるのだろう。しかしこのマウスも朝になると不調をきたす。ホイールのスクロールが機能しなくなるのだ。スケルトンマウスのホイールは機械式だったが、この光学マウスのホイールは光検出式なのだろう。それでホイール部分のわずかな隙間から光が入ることによって機能が停止してしまうのだろう。案の定ホイール部分を手で覆うとまともに動いた。

結論：マウスは夜行性である。

○ピンクのディスプレイ (10月某日)

うちのディスプレイ(某ʃＵＭʃＵＮＧ製17インチ ʃyncMaster753DF)が省電力モードからの復帰時に突然ピンク色に染まった。真っ黒の画面にしても明らかに赤成分だけ余計に出ている。ディスプレイのフロントにある操作パネルをいじっても一向に回復しない。ふたを開けて中を覗いてみようかとも思ったがそれをすると保証が受けられなくなるのであきらめた。何日か色々試したが直る気配がないのでソフマップの修理工場に修理を依頼した(もちろん着払いで)。私の下宿にはディスプレイが1台しかなく、修理に出してしまうとパソコンが使えなくなる。しかし私のパソコンにはAll in Wonderが刺さっている。このビデオカードはテレビチュ

ーナー内蔵でいろんな端子が刺さるようになっている。そこで私はコンピュータ部の部室に置いてある、使ってなさそうなテレビを持ち帰り、接続を試みた。すると、ぼやけているし画面が小さいがきちんと映し出してくれた。しかしこんな環境では文字もろくに読むことが出来ない。仕方なく私は全てのフォントサイズを20くらいに上げ、さらには虫眼鏡ツールを導入してコンピュータを操作することになった。

2週間後、ディスプレイが戻ってきた。どうやら「ガンマ補正」とかいう作業によって見事復活したようだ。岐津氏に聞いた話によるとディスプレイは感電ポイントが多く知らない人がふたを開けると大変危険だそうだ。マウスのような単純なものならふたを開ければ直せるかもしれないがディスプレイはふたを開けると死ぬらしい。

○無料プロバイダ活用法 (11月1日) (ValueNet追悼記念)

最近テレビなどで無料プロバイダのCMを見かけるが、電話料金の高い現状においてはたいへんありがたいことである。私は今年の4月から無料プロバイダを利用している。最初はいろんなプロバイダを試してみたが、結局広告が少ないValueNetに落ち着いた。仮登録でも接続でき、NTT Comunicationのある無料のサービスに登録するだけで会員接続が可能となった。しかし顧客の増加や広告収入が思ったより得られなかったこともあり、2000年9月25日にプロバイダZEROに吸収され、11月1日にサービスは完全に停止した。
今回は無料プロバイダのみを利用した接続環境の作り方を紹介する。

無料プロバイダのサービスには広告・制限がある場合が多い。それらを下に列挙してみる。下に行くほど致命的なものである。

1.接続するとあるサイトに飛ばされる
2.ダイレクトメールが届く
3.接続してから一定時間はリンクボタンを押すとあるサイトに飛ばされる
4.ブラウザーのポップアップウインドウで広告が出る
5.接続ソフトが必要
6.外部のSMTPサーバが使えない
7.画面上部に広告が出る
8.年会費が必要
9.接続時間が月○○時間を超えると従量課金になる
10.入会時にある会社のある契約が必要
11.テレホーダイ・タイムプラスが適用されない
12.テレホーダイ時間は接続不可

まず、ほとんどの無料プロバイダは広告が出る。なぜなら無料プロバイダはほとんどの場合が広告収入で経営されているからである。1・2は大して不便を感じないが、3・6に至っては否応なしに広告を見ることになる。3はポップアップウインドウを閉じるツールを使うことによってある程度は見ないで済むが、6の場合はディスプレイの一部分を完全に占拠してしまうので多少画面が狭くなったと感じるかもしれない。全画面のゲーム(Age of Kingsなど)をプレイするときはこの広告のせいでマウスポイントが合わないので注意が必要だ。4の接続ソフトは細かい設定ができないという点、また、常駐させておく必要があるという点に問題があるがそう大した問題でもないだろう。5の問題はつまりメール受信はできるがメール送信ができないということである。メールを送るにはWebメールかまたはそのプロバイダが用意したメールアドレスを利用するしかないのである。7・8はプロバイダZEROのことであるが、はっきり言って問題外で

ある。無料プロバイダは無料でなければならない。しかも送金にクレジットカードを必要とするために私のようなカードを持たない学生にとってはかえって不便である。9は今は亡きValueNetのことであるが、これも面倒な作業を必要とするのでできれば避けたいものである。10・11は最も致命的な部類に入る項目である。接続料金をできるだけ少なくするために無料プロバイダを利用するのに電話代がかさんでは本末転倒である。

よって、許容される項目は4までである。この条件をクリアする無料プロバイダはいくつかあると思われるが、全ての無料プロバイダを試したわけではないので詳細はわからないが列挙する。

1. FreeLane (項目3,4該当)
2. Auric (項目3,4該当 ただし33.6kbpsまで)
3. freeserve (項目2該当)
4. FREECOM (項目2,4該当)

さて、実際に無料プロバイダで接続してみよう。私は京都に住んでいるのでまず、市内にアクセスポイントの無いfreeserveはあきらめた。freeserveが一番条件は良いのに残念だが、コストの問題があるので仕方が無い。Auricは速度に制限があるので除外。FREECOMはメール関係がややこしい上にDM・広告窓が多いらしいので除外。FreeLaneを利用することに決めた。FreeLaneはサーチエンジンで有名なexcite(http://www.excite.co.jp/)が提供するサービスで、IPアドレスを見る限りではAuricと同じサーバを使っているらしい。設定・サインアップのためのソフトをダウンロード・起動させ、必要事項を書いて送ると即時にアカウントが発行される。勝手にダイヤルアップネットワークにFreeLaneが登録されるので(かなりびびった)、初心者でも簡単である。これで一応接続可能となった。さっそく接続してみるが、最初の30秒はどこに飛んでもexciteのサイトに強制移動を強いられる。これは最初からわかっていたことなので仕方ない。そして気になるポップアップウインドウだが、サイトを移動するたびに広告窓が出る。この広告はClosePopupというツールで消すことができる。だが、これとは別に15分置きくらいに別な種類の小窓が出るようだ。この小窓はなぜかそのツールでは消えてくれなかった。いちいち手で消すかずっと表示させておく必要がある。

テレホーダイ時間の混雑回避法について書いておこう。テレホーダイ適用時間帯は23：00～8：00である。当然23：00～1：00あたりが最も混むのだが、無料プロバイダの場合(一般的なプロバイダもそうかもしれないが)、接続が許される人数に限りがある場合が多い。つまり早い者勝ちである。無料プロバイダが混雑するのは宿命的なものなので悠長に23：00に接続したのでは接続できないことは必至だろう。そこで、テレホーダイ時間帯の20～30分前に接続を開始することを勧める。この時間帯ならばほぼ確実に接続可能である。テレホーダイ適用外の20～30分の分の電話量がバカにならないという人にはタイムプラスに加入するのも一つの手だろう。私はこれに加入しているが多少の得になるようだ。

ある。無料プロバイダは無料でなければならない。しかも送金にクレジットカードを必要とするために私のようなカードを持たない学生にとってはかえって不便である。9は今は亡きValueNetのことであるが、これも面倒な作業を必要とするのでできれば避けたいものである。10・11は最も致命的な部類に入る項目である。接続料金をできるだけ少なくするために無料プロバイダを利用するのに電話代がかさんでは本末転倒である。

よって、許容される項目は4までである。この条件をクリアする無料プロバイダはいくつかあると思われるが、全ての無料プロバイダを試したわけではないので詳細はわからないが列挙する。

1. FreeLane (項目3,4該当)
2. Auric (項目3,4該当 ただし33.6kbpsまで)
3. freeserve (項目2該当)
4. FREECOM (項目2,4該当)

さて、実際に無料プロバイダで接続してみよう。私は京都に住んでいるのでまず、市内にアクセスポイントの無いfreeserveはあきらめた。freeserveが一番条件は良いのに残念だが、コストの問題があるので仕方が無い。Auricは速度に制限があるので除外。FREECOMはメール関係がややこしい上にDM・広告窓が多いらしいので除外。FreeLaneを利用することに決めた。FreeLaneはサーチエンジンで有名なexcite(http://www.excite.co.jp/)が提供するサービスで、IPアドレスを見る限りではAuricと同じサーバを使っているらしい。設定・サインアップのためのソフトをダウンロード・起動させ、必要事項を書いて送ると即時にアカウントが発行される。勝手にダイヤルアップネットワークにFreeLaneが登録されるので(かなりびびった)、初心者でも簡単である。これで一応接続可能となった。さっそく接続してみるが、最初の30秒はどこに飛んでもexciteのサイトに強制移動を強いられる。これは最初からわかっていたことなので仕方ない。そして気になるポップアップウインドウだが、サイトを移動するたびに広告窓が出る。この広告はClosePopupというツールで消すことができる。だが、これとは別に15分置きくらいに別な種類の小窓が出るようだ。この小窓はなぜかそのツールでは消えてくれなかった。いちいち手で消すかずっと表示させておく必要がある。

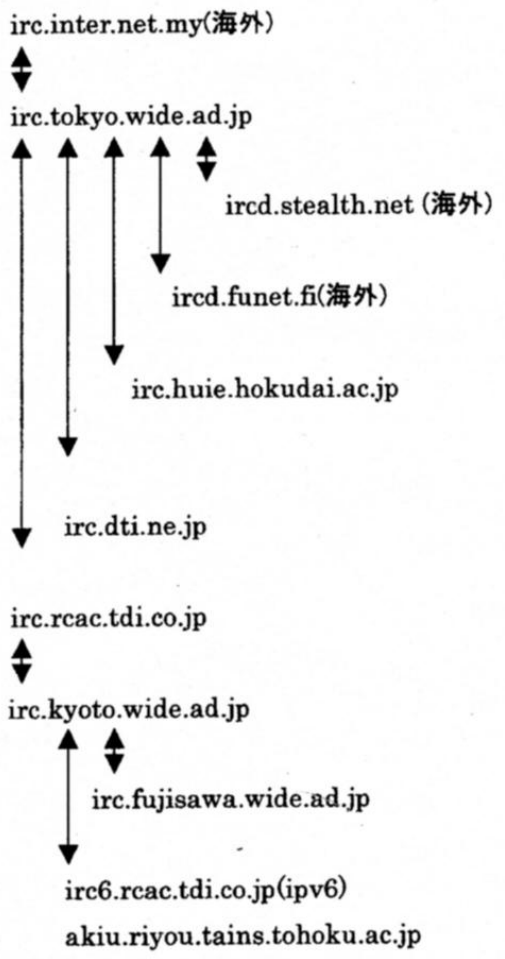
テレホーダイ時間の混雑回避法について書いておこう。テレホーダイ適用時間帯は23：00～8：00である。当然23：00～1：00あたりが最も混むのだが、無料プロバイダの場合(一般的なプロバイダもそうかもしれないが)、接続が許される人数に限りがある場合が多い。つまり早い者勝ちである。無料プロバイダが混雑するのは宿命的なものなので悠長に23：00に接続したのでは接続できないことは必至だろう。そこで、テレホーダイ時間帯の20～30分前に接続を開始することを勧める。この時間帯ならばほぼ確実に接続可能である。テレホーダイ適用外の20～30分の分の電話量がバカにならないという人にはタイムプラスに加入するのも一つの手だろう。私はこれに加入しているが多少の得になるようだ。

IRC(Internet Relay Chat)についての適当な解説

電子情報工学科　2回生　安達　洋明

IRCとはインターネットをつかったチャットシステムの一種です。一つ以上のサーバから構成され、そのサーバ同士が会話などのデータをやりとりすることによってかなりの大人数でのチャットが可能です。
といっても接続してる人全員が同時に話すわけではなくチャンネルと呼ばれるチャットルームのような物があり、そこに分かれてチャット出来ます。

日本のIRCサーバ構成は以下の通りです。IRCNetと接続されています。



このなかのどれかのサーバに接続すれば他のサーバに接続している人とも会話ができるって訳です。
このほかにも一つだけで構成され私的な目的に使用されるサーバや、そういった有志が独立して接続してるIRCネットワークが存在します。

サーバの接続形態は木構造のみが使え、循環的な木構造-網状に接続することはできません。
そのため遠いサーバ(上の図で経由サーバ数が多い)に相手が接続していたり、途中のサーバが重い(回線が混雑している)状態だと遅延が起こりやすくリアルタイムに会話が出来なくなります。そのため一般の IRC ネットを使う場合はチャンネルともにサーバ名が書いてあることが多いです。混雑がひどい状態が続くと通信が全く途絶えてしまいサーバ分断(server split)が起こることがあります。

チャンネルとユーザ

チャンネルに入っているとそのチャンネルに向けて発言されたメッセージを読むことが出来ます。標準の状態ではチャンネルに入らずに発言をすることが出来ます。
発言先をチャンネル名ではなく存在するユーザ名(ニックネーム)にするとその人のみに発言ができ、プライベートなチャットが出来ます。
入ろうとしているチャンネルが存在しなければ参加時に自動的に作られ、誰もいなくなると勝手になくなります。ようするにチャンネル名が重複しなければ誰でも新しいチャンネルを作ることが出来ます。
またチャンネルにはトピックをつけることができ、変更しなければチャンネルが存在する限り保持されます。
ユーザとチャンネルにはモードという属性が存在し、色々な設定が出来ます。

ユーザモード

i	クライアントを不可視(Invisible)とします。 接続ユーザー一覧に出なくなります。
s	サーバの NOTICE(Server notice)メッセージ(サーバからのお知らせ)を受信します。 現在は使われていません。
w	クライアントは WALLOPS メッセージ(全 IRC オペレータへのお知らせ)を受信します。
r	逆引きの失敗や管理上の都合によって制限(restricted)状態になります。(接続時に付き、切断しない限り変更不可)
a	サーバー間で AWAY の設定を行います。AWAY メッセージは付きません。

チャンネル属性

チャンネルモードの内容は以下の通りですが、チャンネル名の頭の記号によってそのチャンネルの属性がまた決まります。

#	通常のチャンネル
!	サーバ分断(server split)にトラブルを避けるため内部的に別のチャンネル名を用いるようにします。
&	独立チャンネルになります。
%	正確にはこの記号のつくチャンネル名は存在しません。 *#channel*.*jp という名前のチャンネルをクライアント側で*%channnel* として扱っているだけです。.*jp をつけると経由 irc サーバが日本の人のみが入れるようになります。 %をサポートするクライアント:irchat、CHOCOA など

+	チャンネルオペレータ、チャンネルモード、トピックが無くなります。

チャンネルモード

o	オペレータ特権を授与/剥奪します。 オペレータはそのチャンネルで特殊コマンド(KICK など)が使用でき、チャンネルモードが変更できます。
p	プライベートチャンネル(Private channel)属性を設定/解除します。 チャンネル外の人の/list や/names や/who でチャンネル名が出なくなります。
s	シークレットチャンネル(Secret channel)属性を設定/解除します。 あらゆる出力から完全にチャンネル名を隠します。
i	インバイトオンリー(Invite-only)属性を設定/解除します。 招待されない人は参加できなくなります。
t	トピックの変更権(Topic settable)をチャンネルオペレータのみに設定/解除します。
n	チャンネル外のクライアントのメッセージ(No message)の受信を許可/不許可します。 チャンネルに入ってない人は発言できなくなります。
m	モデレートチャンネル(Moderated channel:司会つきチャンネル)属性を設定/解除します。 発言が許可されてる人以外は発言できなくなります。
l	チャンネルに参加するクライアントの数を制限(user limit)する。
b	クライアントの侵入を拒む BAN マスク(Ban mask)を設定/解除します。
v	モデレートチャンネル属性のついているチャンネルでの発言権を授与/剥奪します。
k	チャンネルキー(channelkey)(=password)を設定/解除します。
e	b から除外する人を同じようにマスクを設定します。
I	I から除外する人をマスクで設定します。
l	チャンネルに入れる人数を制限します。
O	!が頭に付くチャンネル(!チャンネル)を作った人が持つモードです。これがないと!チャンネルにおいてチャンネルモード+a や+r が付けられません。
r	!チャンネルで全員がチャンネルオペレータでなくなってある程度の時間経過後、ランダムにオペレータ権限を与えます。
a	ニックネームをあかさず匿名で(anonymous)にチャットできるようになります。但し&や!が頭に付くチャンネルにしかつけられません。

※マスクはユーザ名 aho!boke@kitcc.org に対して aho!boke@* のようにワイルドカードで指定します。

そのほか(まとめるのが面倒なので適当)

何気なくクライアントが勝手に使ってるコマンド達

PRIVMSG

普通に発言するときに使われるコマンド。

NOTICE

プログラム的(マクロや自動応答ロボットなど)な反応をするときに使われるコマンド。

マクロや自動応答ロボットはこれに対して反応してはいけない。

これによって無限ループを防ぐ。

CTCP

クライアント同士の通信に使われるコマンド。

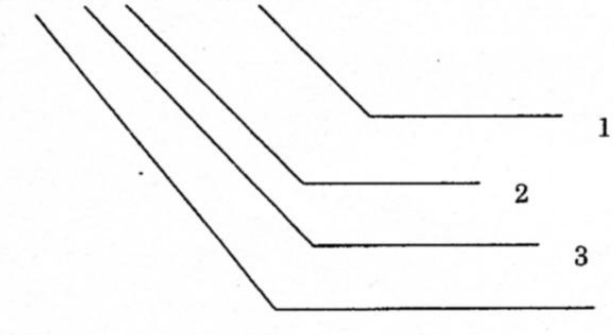
各々のクライアントが勝手に仕様決めて使っていい(クライアント依存)が、PING(任意の数字を送って、送ったときと送り返された時の時間差によりタイムラグを求める)、VERSION(クライアントの情報を返す)、TIME(相手の現地時間を調べる)など基本的なものはほとんどのクライアントで実装されています。

DCC

サーバを介さないクライアント同士の通信に使われるコマンド。CTCP と同じくクライアント依存です。SEND(ファイルを送る)、CHAT(話す)などがあります。

ユーザ名について

aho ! ~ boke @ kitcc.org



1	リモートホスト。
2	Identd(簡易認証サーバ)からの文字列。サーバが無い場合は自分で入力でき、自分のアカウント名を使う場合が多い。
3	Identd の状態と i/I-Line(IRC サーバの管理設定)に入ってるかどうか。 なし…I line with ident/^…I line with OTHER type ident/~…I line, no ident/+…i line with ident/=…i line with OTHER type ident/-…i line, no ident
4	接続時に設定したニックネーム

なんとなくWebサーバ構築

電子情報工学科 2 回生 大宮広義
ohmisan@kitcc.org

平成 12 年 11 月 21 日

1 はじめに

まず、私は FreeBSD を使っているので 本稿は FreeBSD に特化した内容となっている点に注意してください。ちなみに FreeBSD 4.1-Release です。さて 私は普段何をしている人かといいますと、CATV インターネットサービスで高速回線環境にあって、ドメインネームをとって、自宅にサーバ (Web サーバなど) を構築して、自己満足な世界に浸っている人です。自宅にサーバを構築して、まず動いていることを見せるのに手っ取り早いのがそのサーバの Web ページを見せることでしょう。「ほら、これが自宅のサーバやねん」とかいいつつ話も進むわけです。最近は CATV インターネットサービスやフレッツ ISDN(ISDN 回線から NTT 指定の電話番号にかけるとユーザ指定のプロバイダーに接続してくれて、この間の通信料金が時間に関係なくて定額で済むというサービス) などの常時接続サービスが普及しつつあり、個人的に自宅にサーバを構築するといった機会が増えつつあります。そういったわけで、こういう背景からも Web サーバ構築というお題でお話をしようと思います。

2 HTTP サーバについて

「そもそも HTTP サーバというのはなんなの?」素朴な疑問です。表題には Web サーバと書いていますが、正しくは HTTP サーバといいます。Web ブラウジングをされた方なら誰でも Internet Explorer, Netscape Navigator といった Web ブラウザを聞いたことがあると思います。例えば、Web ブラウザ上で http://www.apache.org/httpd.html と打つとそのページがサーバ上に存在すれば、そのページを見ることができます。(ちなみに http://www.apache.org/httpd.html といったものを一般に URL といいます。) つまり Web ブラウザによって「www.apache.org という HTTP サーバにある httpd.html を見せてよ」というリクエストを HTTP サーバ (www.apache.org) に送るわけです。それを受け付けた HTTP サーバはリクエストされたページをその Web ブラウザに返します。これでそのページが見れているわけです。HTTP サーバの仕事は実はこれだけではなく、実にさまざまなことをやっています。とりあえず HTTP サーバとは基本的にはこういったことをしてくれるサーバです。

3　HTTP サーバと Web ブラウザとのやりとり

HTTP サーバというものをもう少し詳しく見ていくことにしましょう。「ガタガタいってないで早く HTTP サーバの設定の仕方を教えろや」という方はこのセクションは読み飛ばしてもらっても構いません。

HTTP サーバは原則としてそのホストの 80 番ポートでリクエストを受け付けます。ポート番号とは役所でいう専用窓口と一緒です。ポート番号は 1 番から 1024 番までが well known port と呼ばれ、いろいろなサービスに割り当てられています。例えば 21 番は FTP(File Transfer Protocol), 119 番は News サーバに割り当てられています。

基本的に 80 番ポートなのですが、設定によって変更することも可能です。しかしながら Web ブラウザはデフォルトで 80 番ポートにリクエストを送るので、変更したところであまりメリットはありません。

さて、HTTP サーバと Web ブラウザは HTTP(Hyper Text Protocol) によって通信します。ここで Protocol というものについて説明する必要があるでしょう。Protocol とは「情報をやりとりする取り決め」とされています。これでは実感がないので例を出すと、Protocol とはサーバにリクエストする時には GET というコマンドを使って、

GET http://www.apache.org/httpd.html

という風な形式で送るんですよ というようなルールのことです。ただそれだけのことです。ですからそのルールを指定する意味で URL の最初に http と書きます。そうです、FTP の場合は URL の最初に ftp と書くわけです。HTTP に話をもどすと、HTTP は比較的簡単なプロトコルで、ただ単に「Web ブラウザからリクエストを送信し、HTTP サーバが適切なレスポンスを返す」ということを必要な回数だけ繰り返すというものです。例えばあるページをみたい時にまず本文 (HTML ソース) を、そしてそのページに画像ファイルがあればそのファイルをひとつづつ送ってくれとサーバ側にリクエストを送信します。具体的に説明すると、例えば www.apache.org に本文、画像 1、画像 2 を含んだページがあった場合、

1. www.apache.org の 80 番ポートに接続
2. そのページをくれとリクエストを送信
3. そのページを取得したら、そのページを解析して追加リクエストの必要性を検証
4. 画像 1 をくれとリクエストを送信
5. 画像 1 を取得、さらに画像 2 をくれとリクエストを送信
6. 画像 2 を取得

これよりさらに Web ブラウザは画像の配置等の作業も行ないますが、いまは割愛します。HTML という言葉を出しましたが、これは Web ページを記述するための言語の一つです。HTML ソースというのは要するに HTML という言語を使って書かれた Web ページの設計文のことです。¡¿でくくられたタグと呼ばれるものを使って記述するのが特徴です。(¡HTML¿, ¡BR¿ など。Web ブラウザでそのページのソースを表示できるので、それを見ればどんなものかはわかると思います) HTML についてのこれ以上の情報は文献がたくさん出回っているので、そちらを参照してください。ここではこれ以上の説明は避けたいと思います。

4 HTTP サーバの動作

さて、HTTP サーバは前述のとおり (基本的に)80 番ポートで Web ブラウザ (一般的にはクライアント) からの接続およびリクエストを常に待ち受けています。Web ブラウザからのリクエストには、通常次のような情報がふくまれています。

・接続先のホスト名 (www.apache.org といったサーバ名)
・リクエストの種類
・求める情報へのパス (/httpd.html といったサーバ名の後につくサーバ上のファイルの位置)
・Web ブラウザが処理できるデータの種類
・プロトコルのヴァージョン (HTTP1.0 か HTTP1.1)

HTTP サーバはこういったリクエストを受け付けると、そのリクエストに対する適切な処理を始めます。

いま http://www.apache.org/httpd.html というリクエストが来た場合を考えます。通常 UNIX 系のファイルシステム同様、/httpd.html はルートディレクトリ (トップのディレクトリ) にある httpd.html というファイルを示しています。HTTP サーバではあらかじめ起点となるルートディレクトリが指定してあり (例えばそのサーバ上の /usr/local/apache/ や /var/www/ など) そこを HTTP サーバのルートディレクトリとしています。(あとで述べますがこれを設定ファイル上で DocumentRoot というオプションで指定します) HTTP サーバはまずそのディレクトリを見に行き、そのファイルがそこにあればそのファイルを返すべきデータとします。いまの場合、HTML で書かれたファイルが指定されていますが、指定されたファイルが CGI スクリプト (Perl という言語で書かれた設計文のようなもの、インターネット上の掲示板や Web チャットは大抵 CGI スクリプトです。) であれば、そのスクリプトを解析、実行します。指定されたファイルの形式によって HTTP サーバが適切な処理を行なっていることがわかると思います。次に送り返すデータを特定できれば後は Web ブラウザ側にそのデータはどんなものかを知らせるためにそのデータの形式の情報を付加します。

5 Apache

HTTP サーバを構築するに当たって、迷わず選ぶのは Apache でしょう。現段階では apache 1.3.14 が最新版 (2000/11/5) となっています。ここからダウンロードできます。

http://www.apache.or.jp/misc/download.htm

インストールはいたって簡単です。

$ tar zxvf apache_1.3.14.tar.gz $ cd apache_1.3.14 $ ./configure
$ make
$ su
# make install

実際には configure のオプションを指定することで、いろいろとできるんですが、非常に多いのでここでは割愛します。

設定ファイルとなる httpd.conf は、/usr/local/apache/conf/ にインストールされます。インストールが終ったので 次は設定ファイルを書きかえます。

$ cd /usr/local/apache/conf/

ここからはお好きなエディターで httpd.conf を編集してください。

ServerRoot “/usr/local/apache”

ここは変える必要ありません。ServerRoot というのは設定ファイルのある conf ディレクトリやログが保存される log ディレクトリがあるディレクトリがある場所を指します。デフォルトで問題ないので ここは変更する必要はありません。

ServerAdmin foo@example.com

ここはエラーが発生した時に自動生成されるページに表示される管理者の電子メールアドレスを指します。ここには自分の持っている電子メールアドレスを指定してください。

DocumentRoot “/usr/local/apache/htdocs”

これは文字どおりその HTTP サーバ上のドキュメントファイルが置かれていルートディレクトリを指定するものです。例えば http://www.apache.org/ などの名前でリクエストされた時、そこの DocumentRoot にある index.html というファイルを返します。つまり http://www.apache.org/ と http://www.apache.org/index.html は等価であると言えます。DocumentRoot の index.html を好きなものに置き換えることで、そのサーバのトップページをお好みに変えることができます。

とりあえずは、これで動きます。ブラウザでプロキシサーバを介さずに http://localhost/ に接続するか、直に telnet localhost 80 をコマンドラインで叩くかして動いているかを確認しましょう。動いてなければ /usr/local/apache/logs/error_log を見て、適切な対処をしてください。

6 httpd.conf の基本的な書式

ここでさらに話を進めていくには、httpd.conf の基本的な書式について知っていなければなりません。基本的に次のような書式で書かれています。

1.
   パラメータの名前　値
2.
   < パラメータの名前　値 >
   ⋮
   ⋮
   </パラメータの名前　値 >

こういった「パラメータの名前」のことを「ディレクティブ」といいます。1 は単にそのディレクティブに対してどんな値をセットするかを指定するものです。一方 2 は、ディレクティブの適応範囲を特定のバーチャルホスト、ディレクトリ、ファイルの範囲内だけに制限するために指定するものです。たとえばここのディレクトリだけに CGI スクリプトを置いて、ここに置かれた CGI スクリプトだけ実行可能にしよう とかいう時にこれで指定するわけです。これを特にブロックディレクティブといいます。ブロックディレクティブは HTML タグに似ていることがわかると思います。これを指定するディレクティブには「VirtualHost」,「Directory」,「Files」,「Location」などがあります。

7 CGI を使えるようにする

CGI というのはそもそも Common Gateway Interface の略で、一種のプロトコルです。HTTP サーバその CGI スクリプトに対するリクエストを受けると、そのスクリプトを起動し大抵 その結果を返します。CGI というのはこのような定義ですから特にプログラムが Perl でないといけないということもありません。そのサーバ上で起動できるプログラム (たとえば C 言語で書かれた実行ファイルなど) でもいいわけです。

さて CGI を使えるようにするために具体的に設定してみましょう。そのためにはモジュールというものの組み込みが必要ですが、デフォルトで組み込まれるようになっているので、特にモジュールに関しては設定する必要はありません。やるべきことは、先ほど説明したディレクティブを設定してあげることです。

/usr/local/apache/conf/httpd.conf にはデフォルトで次のように指定されています。いろんなディレクティブが指定されていますが、今は気にしないでおきましょう。

```
<Directory "/usr/local/apache/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

さていまから /usr/local/apache/cgi-bin に置かれたすべてのファイルを CGI スクリプトとみなすようにします。「えっ？そんなんしていいの？」と思われるかもしれませんが 外からそのディレクトリにどんなファイルが入ってるか見えませんし、それに他のページからその CGI スクリプトへ適切にリンクしてやれば何も問題ありません。これを可能にするディレクティブに SetHandler というものがあります。具体的には、

```
<Directory "/usr/local/apache/cgi-bin">
    AllowOverride None
    Options None
    SetHandler cgi – handler
    Order allow,deny
```

```
    Allow from all
</Directory>
```

と指定してやることで今のことが可能になります。さらに Options というディレクティブを使って CGI スクリプトを実行できるようにしてやります。

```
<Directory "/usr/local/apache/cgi-bin">
    AllowOverride None
    Options +ExecCGI
    SetHandler cgi-handler
    Order allow,deny
    Allow from all
</Directory>
```

ここで気をつけなければならないのは ExecCGI の前の "+" です。/usr/local/apache/conf/httpd.conf 内ではデフォルトで

```
Options Indexes FollowSymLinks MultiViews
```

という具合に Options ディレクティブが指定されています。これはグローバルに指定されているのですべての設定に有効となっています。今、上の設定で

```
Options ExecCGI
```

と書いたとすると、デフォルトで指定されている Options Indexes FollowSymLinks MultiViews が /usr/local/apache/cgi-bin 内ですべて無効になり、Options ExecCGI のみが有効となります。ですから デフォルトの設定を生かしておくために +ExecCGI と書く必要があります。

ここでその他のディレクティブについても説明しておきます。

```
AllowOverride None
```

は .htaccess というファイルによる設定の上書きを禁止するものです。例えば 一般ユーザが自分のホームディレクトリに public_html/ というディレクトリを作って、そのなかで Web ページなどを公開できるのですが、そのディレクトリ内で HTTP サーバ側で AllowOverride All とかしておくと、その中に作ったサブディレクトリ内に .htaccess をうまく置くことでユーザがこのディレクトリは世界のどこからでも見れるようにする、こちらのディレクトリはローカルネット内からしか見せないという細かい設定を行なうことができます。用は HTTP サーバの設定をそのディレクトリ内だけ自分用に書き換えれるわけです。これを応用すれば悪意のある一般ユーザがよろしくないことをすることができるわけです。これを禁止あるいは可能にできます。また管理者が /usr/local/apache/cgi-bin 内の設定を一時的に変えたい時に そこに .htaccess を置いて、設定を上書きすることで HTTP サーバを再起動することなく変更を行なえるという便利さもあります。しかしこの場合リクエストが来るたびにこのファイルが走査されるのでパフォーマンスが低下します。

```
Order allow,deny
Allow from all
```

はどんなホストにも公開するということです。今はこれでいいのでこれ以上の説明は避けます。

あとはそこらから CGI スクリプトを持ってきて、/usr/local/apache/cgi-bin/ に置いて、ブラウザから試してみて下さい。うまく動けば (見れれば) 成功です。

8 SSI を使えるようにする

次に SSI 機能を使えるようにしましょう。SSI というのは Server Side Include の略で、これは CGI と並んでよく利用されるものです。HTML ファイルの中で SSI コマンドと呼ばれるもので記述しておくと、その部分が毎回解析され、その結果が置き換えられて表示されるというものです。例えばリクエストのあった時刻を表示したり、リクエストを送ってきたクライアントによって表示内容を変えたりすることができます。SSI 機能をテストするために /usr/local/apache/ssi/ というディレクトリを作ります。そしてお好きなエディタでそのディレクトリ内に次のような test.shtml というファイルを作って下さい。

```
<HTML>
  <HEAD>
    <TITLE> SSI TEST </TITLE>
  </HEAD>
  <BODY>
    <!--#echo var=''DATE_LOCAL'' -->
  </BODY>
</HTML>
```

ここでは SSI コマンドについての具体的な解説は割愛します。SSI コマンドについては各文献を参考にしてください。

次に具体的な設定に移ります。これは CGI の場合とほとんど同じです。

```
<Directory "/usr/local/apache/ssi">
    AllowOverride None
    Options +Includes
    AddHandler server-parsed .shtml
    AddType text/html .shtml
    Order allow,deny
    Allow from all
</Directory>
```

さて、新しく出てきたディレクティブについて説明します。

```
AddHandler server-parsed .shtml
```

はこのディレクトリ内の .shtml という拡張子を持つファイルすべてを SSI のファイルと見なすようにするものです。CGI の時とはディレクティブが異なることに注意が必要です。もちろん CGI のところで

AddHandler cgi-script .cgi

という指定も可能です。

AddType text/html .shtml

これは .shtml という拡張子を持つファイルを明示的に HTML 形式のテキストファイルであると明示するものです。Web ブラウザ側に .shtml ファイルを HTML 形式であると正しく認識させるために必要です。一般に、

MIME/タイプ

といった具合に示します。例えば JPEG 形式の画像ファイルなら

image/jpeg

といった風になります。

Options +Includes

これは、CGI のときと同様"+" がいります。そしてこれ自体は SSI の機能をこのディレクトリ内で有効にするものです。

実際にブラウザなどで test.shtml を見てみてください。うまく日付が表示されれば成功です。

9 バーチャルホストの設定

1 台のマシンに複数の IP アドレスを割り当てている場合や、一つの IP アドレスに複数のホスト名が割り当てられているとき、リクエストされたホスト名によって見せるページを変えることが可能です。これによってあたかも複数のホストがあるように見せることができます。

9.1 複数の IP アドレスでのバーチャルホスト

1 台のマシンに複数の IP アドレスが割り当てられている場合、IP アドレスごとに違ったページを見せたいときがあります。こういう時には以下のような設定を行なうことでそれが実現できます。いま 1 台のマシンに 10.0.0.1, 10.0.0.2 という IP アドレスが割り当てられていて、それぞれのホスト名が virtual1.example.com, virtual2.example.com であるとします。

```
<VirtualHost 10.0.0.1>
    ServerName virtual1.example.com
    DocumentRoot /usr/local/apache/virtual/virtual1
</VirtualHost>
<VirtualHost 10.0.0.2>
    ServerName virtual2.example.com
    DocumentRoot /usr/local/apache/virtual/virtual2
</VirtualHost>
```

これで IP アドレスによって違うページを見せることができます。またバーチャルホストごとにそれぞれ別個に DocumentRoot を指定できるので、バーチャルホストごとに違った設定を組むこともできます。

9.2 1つのIPアドレスでのバーチャルホスト

次に1台のマシンにIPアドレスが1つ割り当てられていて、そして複数のホスト名がそのマシンに割り当てられている場合の設定方法を紹介します。さっきの上の場合と違うのはNameVirtualHostというディレクティブを指定する必要があることくらいです。

```
NameVirtualHost 10.0.0.1
<VirtualHost 10.0.0.1>
    ServerName virtual1.example.com
    DocumentRoot /usr/local/apache/virtual/virtual1
</VirtualHost>
<VirtualHost 10.0.0.1>
    ServerName virtual2.example.com
    DocumentRoot /usr/local/apache/virtual/virtual2
</VirtualHost>
```

ここでvirtual3.example.comも割り当てられている場合はどうでしょうか。このホスト名でリクエストが来た場合、virtual3.example.comに対する設定が書いてないので、この場合VirtualHostディレクティブではじめに書いたものが有効になります。この場合、virtual1.example.com用の設定が生かされることになります。ですから、一番はじめにデフォルトとなるバーチャルホストを設定しておき、そしてその他のバーチャルホストの設定を書いておくのがベストです。これを考えた設定を以下に示しておきます。/usr/local/apache/conf/httpd.conf内で、バーチャルホストの設定より上に書いたDocumentRootの設定はバーチャルホストの設定により無効になる点に注意して下さい。このことから下に示す設定が必要になるのです。

```
NameVirtualHost 10.0.0.1
<VirtualHost 10.0.0.1>
    ServerName www.example.com
    DocumentRoot /usr/local/apache/htdocs
</VirtualHost>
<VirtualHost 10.0.0.1>
    ServerName virtual1.example.com
    DocumentRoot /usr/local/apache/virtual/virtual1
</VirtualHost>
<VirtualHost 10.0.0.1>
    ServerName virtual2.example.com
    DocumentRoot /usr/local/apache/virtual/virtual2
</VirtualHost>
```

9.3 バーチャルホストを使って一般ユーザのホームページを公開する

これは応用編です。普段ユーザのホームページはhttp://www.example.com/ foo/ (fooというユーザがいたとすると) で見ることができます。一般ユーザのDocumentRootはデフォルトでそのユーザのホームディレクトリのpublic_html/というディレクトリに割り当てられています。

(/usr/home/foo/public_html/ など) ですが、このページを http://foo.example.com/ で見れたらかっこいいですよね。これはいたって簡単で、上の設定をちょいと書き換えれば可能です。

```
<VirtualHost 10.0.0.1>
    ServerName foo.example.com
    DocumentRoot /usr/home/foo/public_html
</VirtualHost>
```

という設定を書き加えればおしまいです。

10 さいごに

さて、いかがだったでしょうか。はっきりいってまだまだ言いたいことはたくさんあるのですが、基本的なことに焦点を絞ったのと、初心者でも Web サーバを構築できるようにという目的としましたので今回はこれで仕方がありません。さらに学習したい方は O'REILLY から出ている Apache に関する書籍を参考にしていただければいいと思います。はじめにも述べましたが、これからは常時接続環境がますます身近なものになっていきますのでこういった Web サーバを一度でも構築しておけば、あとあと役に立つこともあるはずです。このドキュメントを通して新たな道を発見してもらえれば幸いです。

オセロについて

電子情報工学科　2回生　谷尾　元聡

ゲームの手などを、コンピュータに計算させるのにはどうしたらいいのでしょう。人間がオセロなどのゲームをするときにはどのようにしているのでしょうか。勘でするというのは人間の特技ですが、理詰めでやるとすると、頭の中では、ここにおいたらどうなる、こうなる、などと自分の頭の中に盤面を描いて、色々置き換えたりするのだろうと思います。そしてこの展開は不利だ、負ける、こうなったら有利だ、などと色々判断を下し、最も有利になりそうな、あるいはより不利にならなさそうな手を選びます。コンピュータに計算させる場合でも、このような考え方でできます。

オセロでいえば、盤の状態、つぎはどちらの番なのかといったことをまとめて局面といいます。オセロは、オセロに限らずですが、局面を連ねたものだとも考えられます。ある局面Ａである手を打つと別の局面Ｂになる、というのを最初の局面からつなげていくと、ちょうど木の枝が広がっていくような図ができます。オセロでは盤の石は増えていくだけなので、局面が逆戻りすることはありえませんので、枝がもとのほうに戻ることはありません。この図をたどっていって、分岐があればとりあえずそのひとつをたどり、その局面が不利であると判断すれば分岐にもどり、そうでなければもうひとつ進む、ということを繰り返して、しらみつぶしにやれば、計算でよい手を選ぶことができます。

このようにやっていくと、相手の手も読まなければならなくなります。これは相手が、最もよい手を打つと仮定していきます。もし自分が相手の番であったら、一番よいと判断するであろう手を相手が打つと考える訳です。つまり相手の番自分の番関係なく同じように計算していきます。人間同士だとこの辺に駆け引きがあります。

ではどう局面を判断するかということですが、オセロに関しては、選択肢の数、つまり手の数という有力な判断基準になる数値があります。着手個所がなくなってくると、打ちたくない場所にも打たざるを得なくなってしまいます。

また隅をとれば有利になる事が多いというのもあります。逆に相手に隅は与えないほうがよい訳です。

序盤は石が少ないほうがよいという経験則のようなものもあります。石が少ないほうがかえって選択肢が増えるということです。ただしこれには落とし穴があって、少なくしすぎると全滅することもあります。

オセロでは、盤の場所を、上から１～８の番号を、左からアルファベットをふって表現します。d5、e4 に黒があって、d4、e5 に白をおいた状態から始める訳です。例えば最初の状態から f5、d6、c3、f4、e3、f6、g5、e6、e7 のように打っていくと、白が全滅します。このようにして最終的な局面まで読みきれれば、最高の手を選ぶことができる理屈ですが、先を読む手の数を指数に、局面の選択肢の平均数を適当に求めて底にして、大体計算の量を見積もってみると、１０手、２０手となれば、指数関数的に増えていきますので、とんでもない数の計算をしないといけなくなります。

といったことを踏まえて、思考ルーチンをつくれば、結構つよいのができるかもしれません。といったところで書くこともなくなってきましたのでこれで終わります。お粗末でした。

（参考文献など）

早わかり図解オセロ　日東書院　谷田邦彦
Making of Othello　www.yo.rim.or.jp

Javaとは？(超初級編)

電子情報工学科二回 野川 博司

平成 14 年 10 月 23 日

1　はじめに

Java は、Sun Microsystems 社が開発した、オブジェクト指向プログラミングです。Java 言語は、C++をお手本としてはいますが、コンパクトで、シンプルで、ソースレベルとバイナリレベルの両方において、プラットフォームや OS の違いを超えてポータブルであるように設計されています。

2　Java はプラットフォーム独立

Java が他のプログラミング言語と比べて優っている点のうちでも特に重要なのは、（ソースレベル、バイナリレベルの双方において）プラットフォーム独立だということです。ソースレベルで見ると、Java の基本データ型は、すべての開発プラットフォームにおいて一貫した大きさを持ちます。また、すべての Java 環境に共通した Java の基本クラスライブラリを使うことで、あるプラットフォームで作成したコードを移植先のプラットフォームで動くように書き直す必要もなくなり、プラットフォームをまたがって移せるようなコードを簡単に作成できます。しかも、Java のバイナリファイルもプラットフォーム独立であり、ソースコードを再コンパイルする必要なく、複数のプラットフォームで実行できます。どのようにして、そんなことができるのかと言うと Java のバイナリファイルはバイトコードと呼ばれる形式である、ということがポイントです。通常 C やその他多くの言語では、ソースプログラムをコンパイルすると、コンパイラがソースプログラムを機械語に変換します。それらの命令は、そのコンピュータが使っている CPU に固有のものです。だから、そのプログラムを別のシステム上でも使いたければソースコードに戻って、移植先のシステム用のコンパイラを使って再コンパイルしなければなりません。Java でプログラムを書くと、様子が違ってきます。Java の開発環境には、Java コンパイラと Java インタプリタという、2 つの要素があります。Java コンパイラはプログラムを受け取ると、ソースファイルから機械コードを生成するのではなく、バイトコードを生成します。Java プログラ

ムを実行するには、バイトコードインタプリタと呼ばれるプログラムを起動し、インタプリタがバイトコードの各命令を順番に実行していきます。何故バイトコードインタプリタという段階を余計に設けたのだろうか？

Java プログラムをバイトコード形式にすることで、プログラムの実行は特定のシステムに限定されず、Java インタプリタが使える限りは、どのプラットフォーム、OS、ウィンドウシステム上でも、プログラムが実行できることになります。バイトコードの不利な点は実行速度です。実行できるシステムを限定したプログラムは、コンパイル対象のハードウェア上で直接実行できるため、インタプリタが処理する Java のバイトコードよりかなり高速です。

3　Java はオブジェクト指向

Java はそのオブジェクト指向機能の多くを、お手本とした C++から引き継いでいますが、他のオブジェクト指向言語から持ってきた概念も多くあります。ほとんどのオブジェクト指向言語と同じく、Java にも基本データ型、入出力、その他のユーティリティ機能を提供するクラスライブラリ一式があります。これらの基本クラス群は、JDK の一部です。JDK には、ネットワーク機能、一般的なインターネットプロトコル、ユーザインタフェースツールキットの各機能を提供するクラスが含まれています。これらのクラスライブラリは Java で書かれているので、全ての Java アプリケーション同様に、プラットフォームをまたがってポータブルです。

4　Java はシンプル

Java の設計目標には、コンパクトかつ単純であり、したがって書きやすく、コンパイルも簡単で、デバッグも容易であること、そして何より学ぶのがやさしいということがあげられています。Java は C、C++を手本にしていて、文法やオブジェクト指向の機構は大部分 C++のものを拝借しています。しかし Java は見かけ上、C や C++に似ていますが、C と C++における複雑な部分は大体 Java では取り除かれています。そのため Java はシンプルなのですが、そのために Java の強力さが犠牲になっているわけではありません。Java にはポインタやポインタ演算機能はありません。（参照はありますが）文字列と配列は、通常のオブジェクトとして扱われます。また、Java にはガーベジコレクションがあり、動的に確保したメモリ領域は、参照されなくなると自動的に回収されます。C や C++にあるような malloc や free はありません。

5　アプレットとアプリケーション

Java アプリケーションには、大別するとアプレットとスタンドアロンアプリケーションに分類できます。アプレットとは、WWW 経由で読み手のマシンの web ブラウザ上にダウンロードされ実行される Java プログラムのことです。アプレットの実行は Java 機能を持つブラウザによってなされます。スタンドアロンアプリケーションは、Java 言語で書かれたもっと一般的なプログラムです。アプリケーションの実行にはブラウザは必要ありません。実際、従来のプログラム言語で普通に作ってきたようなアプリケーションを Java でも作ることができます。

6　終わりに

とまあ、ここまで Java の目標とするところや基本的な特徴を述べましたが、僕もまだ Java を勉強中なので、たいしたことが書けませんでした。（本当に触りだけでした、、、、、）でも、これを読んで Java に少しでも興味や関心を持ってくれれば、幸いです。

HTTPでファイルをGETしよう！

電子情報工学科 2 回生 横川 龍雄

はじめに

最近は Web 関連のアプリケーションが多く出回っています。Web 巡回ソフトなどは非常に便利なものです。ここではそういったアプリケーションが使用している HTTP(Hyper Text Transfer Protocol) の簡単な説明しつつ、実際に HTTP を使用してファイルを取得する簡単なプログラムを作るまでの話をさせていただきます。なお、一般的なアプリケーションでは HTTP/1.1 を使用しているものがほとんどですが、ここでは HTTP/1.0 の説明をさせていただきます。また、ここでのプログラムは FreeBSD 上で作成する事を前提とします。

サーバーに接続しよう

さて、HTTP を使用するにもまずサーバーに接続しなければなりません。他のマシン等と通信をするためにはソケットを使用します。具体的には以下のようなコードを使用しました。

```
/* サーバーへの接続 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int                 s, port = 80;
struct sockaddr_in sockaddr;
struct hostent     *host;

s = socket(AF_INET, SOCK_STREAM, 0);
if(s == -1){
  return -1;
}

host = gethostbyname("www.foo.org");
if(host == NULL){
  printf("host name lookup failure!\n");
  return -1;
}

bzero(&sockaddr,sizeof(sockaddr));
sockaddr.sin_family = AF_INET;
sockaddr.sin_port   = htons(port);
bcopy(host->h_addr,&sockaddr.sin_addr,host->h_length);

if(connect(s, (struct sockaddr *)&sockaddr, sizeof(sockaddr))){
  printf("サーバー%s に接続できませんでした\n", host_name);
  return -1;
}
```

まず、*socket* システムコールを呼出して通信を行なうためのエンドポイントを作成します。次に *gethostbyname* を呼出しホストエントリを得て、*sockaddr_in* 構造体 *sockaddr* にアドレスファミリ、接続先のポート番号、*gethostbyname* で得られたサーバーのアドレスを格納します。そして *connect* システムコールを呼出してサーバーとの接続を確立します。私はソケットのことはまだよく理解していないので、詳細はソケットについて書かれた文献を参照して下さい。

リクエストを送ろう

接続も済んだのでいよいよ HTTP の出番です。HTTP のメッセージはクライアントからのリクエストとサーバーのレスポンスから成り立っています。HTTP/0.9 においてはリクエストは 1 行のみでした。また、レスポンスでは URI で指定されるリソースそのものが送られてきました。これに対し HTTP/1.0 においてはリクエストでは Request-Line、レスポンスでは Status-Line と呼ばれる 1 行と複数行のヘッダから成り立っています。

それではサーバーにリクエストを送ってみましょう。ここでは以下のようなコードを使用しました。

```
#include <stdio.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>

char wkday[][4] = {
  "Sun",  "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};

char month[][4] = {
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",
};

char       temp[1024];

struct stat sb;
struct tm   *tm;
time_t      tlc;

/* リクエストの送信 */

sprintf(temp, "%s /%s HTTP/%d.%d\r\n", "GET", "index.html", 1, 0);
write(s, temp, strlen(temp));

tlc = time(NULL);
tm  = gmtime(&tlc);

sprintf(temp, "Date: %3s, %02d %s %4d %02d:%02d:%02d GMT\r\n",
        wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
        tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);
write(s, temp, strlen(temp));

sprintf(temp, "User-Agent: %s/%d.%d\r\n", "hogehoge", 0, 0);
write(s, temp, strlen(temp));

stat("foo.html", &sb);
tm  = gmtime(&sb.st_mtime);

sprintf(temp, "If-Modified-Since: %3s, %02d %s %4d %02d:%02d:%02d GMT\r\n",
        wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
        tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);
write(s, temp, strlen(temp));

sprintf(temp, "\r\n");
write(s, temp, strlen(temp));

/* リクエストの送信終了 */
```

ここで使用している変数 *s* はさきほどサーバーへの接続に使用したものと同じです。相手サーバーへリクエストを送るために *write* システムコールを使用して通常のファイルへの入出力と同じように書き込みを行なっていますが、こういった事ができるソケットは非常に便利な物です。それでは順をおってプログラムを見ていきましょう。

```
sprintf(temp, "%s /%s HTTP/%d.%d\r\n", "GET", "index.html", 1, 0);
```

```
write(s, temp, strlen(temp));
```

ここではリクエストラインを変数 *temp* に格納してサーバーへ転送しています。リクエストラインはメソッド,URI,HTTP のバージョンのそれぞれがスペースで区切り最後に CRLF(”\r\n”) を付けます。メソッドには GET や HEAD 等があります。ここではリソースの転送を要求する GET を使用します。ちなみに HEAD はリソースの転送を行なわないという事以外は GET と同じです (余談ですが、某有名サイトのサーバーでは HEAD でも GET と同じようにリソースを送ってきました:-P)。URI の部分にはそのサーバーのトップからの相対 URI で指定しています。接続しているサーバーがプロキシサーバーであるときは URI は絶対 URI で指定しなければなりません。HTTP のバージョンには HTTP/の後にメジャーバージョンとマイナーバージョンをピリオドで区切って指定します。こうして出来上がったリクエストラインは以下のようになります。

```
GET /index.html HTTP/1.0
```

次の部分はジェネラルヘッダと呼ばれるリクエスト、レスポンスともに共通するヘッダフィールドです。

```
tlc = time(NULL);
tm  = gmtime(&tlc);

sprintf(temp, "Date: %3s, %02d %s %4d %02d:%02d:%02d GMT\r\n",
        wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
        tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);
write(s, temp, strlen(temp));
```

ジェネラルヘッダは複数ありますが、ここでは Date ヘッダしか使用していません。Date ヘッダは Date: の後に RFC 1123 で定義される日付フォーマットが続き CRLF で終ります、送る日付はリクエストを発信した日付を表します。具体的には以下のようになります。

```
Date: Fri, 10 Nov 2000 02:22:07 GMT
```

この次の部分はリクエストヘッダと呼ばれるクライアントがサーバーに対して付加的な情報を送るために用いられるヘッダーフィールドです。

```
sprintf(temp, "User-Agent: %s/%d.%d\r\n", "hogehoge", 0, 0);
write(s, temp, strlen(temp));

stat("foo.html", &sb);
tm  = gmtime(&sb.st_mtime);

sprintf(temp, "If-Modified-Since: %3s, %02d %s %4d %02d:%02d:%02d GMT\r\n",
        wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
        tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);
write(s, temp, strlen(temp));
```

リクエストヘッダも複数あるのですがここでは User-Agent と If-Modified-Since しか使用していません。このうち User-Agent ヘッダはリクエストを発行したクライアントプログムの情報を指定します。書式としてはクライアントプログラムの名前とバージョンをスラッシュで区切り、バージョンはメジャーバージョンとマイナーバージョンをピリオドで区切り、最後は CRLF をつけます。上の例でいいますと以下のようになります。

```
User-Agent: hogehoge/0.0
```

If-Modified-Since ヘッダが指定された GET は条件つき GET となります。ここでは stat システムコールを用いてファイルが最後に更新された日付を得てそれを指定していますが指定した日付より後にリソースが更新されていなければサーバーは”304 Not Modified”というステータスを返しリソースの転送を行ないません。

最後に CRLF のみの行をサーバーに送っていますが、これがリクエストメッセージの終了を意味します。

レスポンスを受け取ろう

リクエストも送ったので次はレスポンスを受け取ります。こちらもリクエストメッセージと同じく CRLF で終る複数の行から成り立っています。ここでは以下の行のような関数で 1 行ずつ取り出しています。

```
int
get_response_line(s, buf, size)
    int  s;
    char *buf;
    int  size;
{
  int  ptr = 0, newline = 0;
  char c;

  while(ptr < size - 1){
    read(s, &c, 1);

    if(c == '\n'){
      buf[ptr]='\0';
      break;
    }else if(c != '\r'){
      buf[ptr++] = c;
    }
  }

  return ptr;
}
```

ここで CR が読み捨てられるようになっていますが、これは必ずしも全てのサーバーが CRLF を返してはくれないからです。では、実際にレスポンスを受け取りリソースを取得してみましょう。ここでは以下のようなコードを使用しました。

```
int i;

i = get_response_line(s, temp, 1024);
printf("%s\n", temp);

if(strncmp(temp, "HTTP", strlen("HTTP")) != 0){
  printf("HTTP サーバーではありません\n");
  return 0;
}else{
  int major, minor;
  int status;

  sscanf(temp, "HTTP/%d.%d %3d", &major, &minor, &status);

  if(status >= 300){
    printf("status code: %d\n", status);
    return 0;
  }
}

while(1){
  i = get_response_line(s, temp, 1024);

  if(i > 0)  printf("%s\n", temp);
  else       break;
}

fp = fopen("foo.html", "w");
if(fp == NULL) return 0;

printf("\n ファイルのダウンロード開始\n");

while(1){
```

```
    i = read(s, temp, 1024);

    if(i > 0)  fwrite(temp, i, 1, fp);
    else       break;
  }

  fclose(fp);

  close(s);

  printf("ダウンロード完了\n");
```

さて、複数行で成り立っているレスポンスメッセージですがここでは手抜きしてレスポンスラインしか見ていません。ここでステータスコードが 300 以上だとプログラムの実行を終了しています。CRLF のみの行がレスポンスメッセージの終りを示すので、ここでは読みとった文字数が 0 以下の時にループから抜けるようにしています。レスポンスメッセージが終れば後はリソースそのものが送られてきますので、それをファイルに保存します。

最後に

いままでのコードを使って作成した HTTP を使ってファイルを取得するプログラムを載せておきます。あまり、参考にならなかったかも知れませんがこれを読んで HTTP に興味をもっていただいたら幸い (?) です。

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

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <netdb.h>

char wkday[][4] = {
  "Sun",  "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};

char month[][4] = {
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",
};

int
get_response_line(s, buf, size)
     int  s;
     char *buf;
     int  size;
{
  int  ptr = 0, newline = 0;
  char c;

  while(ptr < size - 1){
    read(s, &c, 1);

    if(c == '\n'){
      buf[ptr]='\0';
      break;
    }else if(c != '\r'){
      buf[ptr++] = c;
    }
  }

  return ptr;
}
```

```
int
main(argc, argv)
     int  argc;
     char **argv;
{
  struct sockaddr_in sockaddr;
  struct hostent     *host;
  int                s,port = 80;

  struct stat sb;
  struct tm   *tm;
  time_t      tlc;

  char       *host_name = NULL;
  char       *data_name = NULL;
  char       *port_num  = NULL;

  char       *url;
  char       *fname;

  FILE       *fp;

  int        i,ims = 1;
  char       *p;
  char       temp[1024];

  if(argc < 3){
    printf("too few arguments\n");
    return 0;
  }

  fname = argv[--argc];

  if((fp = fopen(fname, "r")) == NULL){
    ims = 0;
  }

  url   = argv[--argc];

  while(argc > 0){
    if(strcmp(argv[argc], "-f")==0){
      ims = 0;
    }
    argc--;
  }

  if(strncmp(url, "http://", strlen("http://")) != 0){
    printf("処理できない URL です\n");
    return 0;
  }

  host_name = url + strlen("http://");

  p = strchr(host_name, '/');
  if(p != NULL){
    *p = '\0';

    if(*(p+1) != '\0') data_name = p + 1;
    else               data_name = "\0";
  }else{
    data_name = "\0";
  }

  p = strchr(host_name, ':');
  if(p != NULL){
    *p = '\0';

    if(*(p+1) != '\0') port_num = p + 1;
    else               port_num = NULL;
  }
```

```
if(port_num != NULL) port = atoi(port_num);

/* サーバーへの接続 */

s = socket(PF_INET, SOCK_STREAM, 0);
if(s == -1){
  printf("socket failed!\n");
  return -1;
}

host = gethostbyname(host_name);
if(host == NULL){
  printf("host name lookup failure!\n");
  return -1;
}

bzero(&sockaddr,sizeof(sockaddr));
sockaddr.sin_family = AF_INET;
sockaddr.sin_port   = htons(port);
bcopy(host->h_addr,&sockaddr.sin_addr,host->h_length);

printf("ホスト%s に接続中\n",host_name);

if(connect(s, (struct sockaddr *)&sockaddr, sizeof(sockaddr))){
  printf("ホスト%s に接続できませんでした\n", host_name);
  return -1;
}

/* ここから GET メソッドの処理 */

sprintf(temp, "%s /%s HTTP/%d.%d\r\n", "GET", data_name, 1, 0);
write(s, temp, strlen(temp));
printf("%s", temp);

tlc = time(NULL);
tm  = gmtime(&tlc);

sprintf(temp, "Date: %3s, %02d %s %4d %02d:%02d:%02d GMT\r\n",
wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);

write(s, temp, strlen(temp));
printf("%s", temp);

sprintf(temp, "User-Agent: %s/%d.%d\r\n", "hogehoge", 0, 0);
write(s, temp, strlen(temp));
printf("%s", temp);

if(ims){
  stat(fname, &sb);
  tm  = gmtime(&sb.st_mtime);

  sprintf(temp, "If-Modified-Since: %3s, %02d %s %4d %02d:%02d:%02d GMT\r\n",
  wkday[tm->tm_wday], tm->tm_mday, month[tm->tm_mon],
  tm->tm_year+1900, tm->tm_hour, tm->tm_min, tm->tm_sec);
  write(s, temp, strlen(temp));
  printf("%s", temp);
}

sprintf(temp, "\r\n");
write(s, temp, strlen(temp));
printf("%s", temp);

i = get_response_line(s, temp, 1024);
printf("%s\n", temp);

if(strncmp(temp, "HTTP", strlen("HTTP")) != 0){
  printf("HTTP サーバーではありません\n");
  return 0;
```

```
  }else{
    int major, minor;
    int status;

    sscanf(temp, "HTTP/%d.%d %3d", &major, &minor, &status);

    if(status >= 300){
      printf("status code: %d\n", status);
      return 0;
    }
  }

  while(1){
    i = get_response_line(s, temp, 1024);

    if(i > 0)  printf("%s\n", temp);
    else       break;
  }

  fp = fopen(fname, "w");
  if(fp == NULL) return 0;

  printf("\n ファイルのダウンロード開始\n");

  while(1){
    i = read(s, temp, 1024);

    if(i > 0)  fwrite(temp, i, 1, fp);
    else       break;
  }

  fclose(fp);

  close(s);

  printf("ダウンロード完了\n");
}
```

rogueって何?

機械システム工学科 2 回生 米田　裕

注:ここに登場する人物はとあるゲームのキャラクターです。そのゲームを知っている人はそのキャラクターが喋っているのだと思いこむことで少しぐらいは楽しめるかもしれません。あと、そのゲームを知らない人のためにごく簡単に人物紹介をしておきます。
神奈…正式な名前は神奈備命、高貴な身分の女の子、しかし言葉使いが少しおかしい。
柳也…神奈の随人、正八位衛門大志であり剣の達人。
裏葉…神奈に付き従う女官のひとり。感が鋭くまた非常に豊富な知識の持ち主。

発端

神奈:柳也どの、何をしておるのだ?
柳也:神奈か、ちょうど今 rogue をやってるんだ。
神奈:ろーぐ?ろーぐとは何だ?
柳也:何だ知らないのか?rogue というのはマップ自動生成型のテキスト RPG だ。
神奈:まっぷじどうせいせいがたてきすとあーるぴーじー?何の事だそれは?
柳也:まあ説明するより実際にやってみたほうがいいな。ほら、やってみろ。
神奈:うむ、やり方を教えるがよいぞ。
柳也:わかったわかった。
神奈:……
柳也:……
神奈:うがあっ、また死んでしまったではないかっ!!!
柳也:おまえって本当にゲーム下手だな。
神奈:それにしてもよくできておるのう。どうなっておるのだ?
柳也:どうなっておるのだと言われてもなぁ…
裏葉:それならば rogue の source を見てみればよろしいのではありませんか?
柳也:うわっ、裏葉!!!
神奈:いつからそこにいたのだ?
裏葉:神奈さまが rogue をおやりになっていらっしゃった頃ぐらいからでしょうか。
神奈:まったく気がつかなかったぞ…
柳也:あいかわらず気配をまったく感じさせないな…
裏葉:そんなことより柳也さま、神奈さまに rogue の source を見せてさしあげたらどうでございましょう。
柳也:しかしな裏葉、神奈が rogue の source を見てそれを理解できるとはとても思えないぞ。
裏葉:それならば柳也さまが御説明してさしあげればよろしいではありませんか。
柳也:それなら何もわざわざ俺が説明しなくても裏葉が説明してやったらいいじゃないか。
裏葉:わたくしは柳也さまが神奈さまに御説明なさるのをおそばで拝聴させていただきます。
柳也:おまえなあ…
神奈:どちらでもよいから説明してくれるのなら早くするがよいぞ。
裏葉:ほらほら柳也さま、お早く。

柳也:しょうがないな…

説明

柳也:じゃあまずはこれを見てくれ。

```
#ifdef WONDERSWAN
#include <sys/bios.h>
#endif
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "rogue.h"
#ifdef WONDERSWAN
#include "vt.h"
#endif

extern short party_room;
#ifndef ORIGINAL
extern char far *nick_name;
#endif

main(argc, argv)
int argc;
char *argv[];
{
#ifndef ORIGINAL
    int first = 1;
    static char buf[80];
#endif
#ifdef WONDERSWAN
    text_screen_init();
#endif /*WONDERSWAN*/
    if (init(argc, argv)) { /* restored game */
        goto PL;
    }
#ifdef WONDERSWAN
    vt_center_charadr(0, 0);
#endif /*WONDERSWAN*/
    for (;;) {
        clear_level();
```

```
        make_level();
        put_objects();
        put_stairs();
        add_traps();
        put_mons();
        put_player(party_room);
        print_stats(STAT_ALL);
#ifdef WONDERSWAN
        vt_center_charadr(rogue.col, rogue.row);
#endif /*WONDERSWAN*/

#ifndef ORIGINAL
        if (first) {
            sprintf(buf,
#ifdef JAPAN
                "やあ、%s。運命の洞窟へようこそ...",
#else
                "Hello %s, welcome to the Dungeons of Doom...",
#endif
#ifdef WONDERSWAN
                md_gln());
#else
                nick_name);
#endif /*WONDERSWAN*/
            message(buf, 0);
        }
#endif
    PL:
        first = 0;
        if (play_level())
            break;
        free_stuff(&level_objects);
        free_stuff(&level_monsters);
    }
    return (0);
}
```

神奈:なんだこれは?
裏葉:これは rogue の source の中の main.c でございますね。
柳也:そうだ。さすが裏葉だな。
裏葉:しかしそれにしては少々短いように感じるのですが…
柳也:ああ、実は main() 以外の関数は省いてあるんだ。あと rogue のオリジナルの source は見つ

けられなかったので、今回は WonderWitch についている rogue の source を用いた。
神奈:何が書いてあるのかさっぱりわからんぞ。
柳也:rogue は C 言語で書かれているからな。
神奈:C 言語とはなんだ?何かのまじないか?
裏葉:C 言語というのはコンピュータ言語のうちの一つでございます。
柳也:C 言語のことから説明してやってもいいんだがそんなことをしていると時間がいくらあっても足りないからな、今日は rogue のプログラムががどんな動きをしているかだけを簡単に見ていくことにしよう。C 言語のことはまた時間のあるときにゆっくり説明してやるよ。だから今日はわからない言葉とかが出てきてもあまり気にするな。
神奈:そうか、すまないな。
柳也:じゃあとりあえず for(;;){} で囲まれた部分を見てくれ。rogue を普通に遊んでいる間は rogue のプログラムはこの部分の処理を繰り返すんだ。その部分の最初に clear_level(); と書いてあるだろう。まずはそこから説明していくことにしよう。
神奈:うむ、わかった。
柳也:C プログラムは関数と変数からなっているんだが clear_level(); は関数で、ここでは clear_level(); という名前の関数を呼び出しているんだ。で、その clear_level(); という関数は level.c にあるんだが…
神奈:ここでは何をしているのだ?
裏葉:いろんな設定を初期化しているようでございますね。
柳也:そうだ、マップ上の部屋の数とか、その部屋ごとのドアの数とか、罠の数と種類とか、画面に新しくマップを作るために必要な設定を初期化しているんだ。
神奈:ということはこの後新しいマップを作っていくわけだな。
柳也:そういうことだな。
神奈:では説明を続けるがよいぞ。
柳也:ああ、じゃあ次の関数を見てみようか。make_level(); という関数を呼び出しているな。
神奈:この関数は何をしておるのだ?
柳也:この関数も clear_level(); と一緒で level.c にあって、ここで新しくマップを作っているんだ。
神奈:マップだけなのか?
柳也:いや、実はこのゲームの目的はイェンダーの魔除けというのを取ってくるのが目的なんだが、一定の階まで進んでなおかつイェンダーの魔除けを持っていない場合、そのイェンダーの魔除けをマップのどこかに配置するようになっていて、それもここで行っている。
神奈:それ以外のあいてむとか罠とかはどうなっておるのだ?
柳也:それはこれから説明するんだ。次に put_objects(); という関数があるだろう。
裏葉:名前から察するに新しく作ったマップにアイテムを配置するための関数…といったところでございましょうか。
柳也:そう…なんだが、先に言うなよ裏葉。
柳也:…まさかおまえわかってて言ったんじゃないだろうな?
裏葉:めっそうもございません。わたくしがなぜ柳也さまの御説明の邪魔をしなければならないのですか。
柳也:そうか?それにしては顔がやけに嬉しそうなんだが…
神奈:こら、余を無視して話をするでない。結局 put_objects(); ではなにをしておるのだ。
柳也:さっき裏葉が言ってたじゃないか、新しく作ったマップにアイテムを配置しているんだ。
神奈:ではその次の put_stairs(); ではなにをしておるのだ?

柳也:put_stairs(); ではマップに階段を配置しているんだ。
神奈:そうか、階段がなければ先にも進めぬし後にも戻れんからな。
裏葉:やはり階段は通路には置かれないように工夫している様でございますね。
神奈:うむ、通路に階段があるマップなど見たことがないからな。
柳也:あと、put_objects(); も put_stairs(); も関数本体は object.c というところにあるんだ。
神奈:clear_level(); や make_level(); とは違う場所にあるのじゃな。
柳也:そうだ。まあ put_stairs(); に関してはこんなところか。
神奈:うむ、次に進むがよいぞ。
柳也:次は add_traps(); だな。これは名前の通り、罠の数と種類を決定して配置しているんだ。
裏葉:今どのぐらいの階にいるかによって罠の数が変わるようになっているようでございますね。
柳也:そうだな。ちゃんとそうやって階が進むごとに難しくなるようになっているみたいだな。
神奈:ずっと同じでは味気ないからな。
柳也:あと add_traps(); は trap.c というところにあるんだ。
柳也:で、その次が put_mons(); か。これも名前の通りで、モンスターの数と種類を決定してマップに配置しているんだ。
裏葉:けれども add_traps(); と違って、今どのぐらいの階にいるかによってモンスターの数が変わったりはしないようでございますね。
柳也:そのかわりモンスターの種類は変わるようだな。
柳也:ちなみに put_mons(); の場所は monster.c だ。
神奈:次はこの put_player(party_room); というやつか。
裏葉:その通りでございます。この関数は名前から察するに…
柳也:キャラクターをマップに配置しているんだ。
裏葉:……
柳也:どうした裏葉?
裏葉:最後まで言わさせていただけないのですね。(泣)
柳也:おまえが言うと絶対にあててしまうからな。
裏葉:それにしても、必ず階段の上に配置されるというわけではないようでございますね。
神奈:なぜだ?階段の上に配置しておくようにしておけばよいのではないのか?
柳也:いや、階段の上だけじゃだめなんだ。罠とかで階段以外の場所に配置されることもあるだろう。
神奈:そうか、なるほどの。
柳也:で、配置された場所に応じてその周りが見えるようになるわけだ。
神奈:そういえば最初からマップすべてが見えるようになっているわけではなかったな。
柳也:そうだ。それと put_player(); は level.c にある。
神奈:clear_level(); や make_level(); があったのと同じところだな。
柳也:そういうことだ。
柳也:次は print_stats(STAT_ALL); という関数なんだが、これはマップ部分以外の、例えばステータスなどを表示するための関数だ。
神奈:マップの下に表示されているあれか。
柳也:そう、それだ。その部分を表示しなおしているんだ。
裏葉:この関数は message.c というところにございます。
柳也:どうして知ってるんだ?
裏葉:柳也さまが神奈さまに御説明していらした間に拝見させていただきました。

柳也:本当か?
裏葉:本当でございますとも。
柳也:まあいいか。じゃあ、次は少し飛ばして play_level(); というところを見てくれるか。
神奈:なぜ少し飛ばすのだ?
柳也:その飛ばした部分はゲームの処理にはあまり関係がないんだ。
神奈:そうか、まあおぬしの好きなようにするがよい。
柳也:ああ。で play_level(); なんだが、その前に今までの関数を見てきて何か気付いたこととかはないか?
裏葉:今までの関数はすべてマップを書きかえるときのもので、肝心のキャラの行動や移動などに関することには一切触れてはございませんね。
柳也:そうだ、そういったキャラの行動や移動に関することをここで処理しているんだ。この関数は play.c にあるんだが、その関数本体を一度見てくれ。

```
boolean
play_level()
{
    short ch, cmd, oldcmd;
    int count;

    cmd = '.';
    for (;;) {
        if (rogue.hp_current <= 0)
            return 1;

        interrupted = 0;
        if (hit_message[0]) {
            message(hit_message, 1);
            hit_message[0] = 0;
        }
        if (trap_door) {
            trap_door = 0;
            return 0;
        }
        move(rogue.row, rogue.col);
        refresh();
        oldcmd = cmd;
        ch = cmd = rgetchar();
        check_message();
        count = 0;
CH:
        switch(ch) {
        case '.':
            rest((count > 0) ? count : 1);
```

```
            break;
        case 's':
            search(((count > 0) ? count : 1), 0);
            break;
        case 'i':
            inventory(&rogue.pack, ALL_OBJECTS);
            break;
#ifndef WONDERSWAN
        case 'f':
            fight(0);
            break;
        case 'F':
            fight(1);
            break;
#endif /* WONDERSWAN */
        case 'h':
        case 'j':
        case 'k':
        case 'l':
        case 'y':
        case 'u':
        case 'n':
        case 'b':
            (void) one_move_rogue(ch, 1);
            break;
        case 'H':
        case 'J':
        case 'K':
        case 'L':
        case 'B':
        case 'Y':
        case 'U':
        case 'N':
#ifndef WONDERSWAN
        case CTRL('H'):
        case CTRL('J'):
        case CTRL('K'):
        case CTRL('L'):
        case CTRL('Y'):
        case CTRL('U'):
        case CTRL('N'):
        case CTRL('B'):
#endif /* WONDERSWAN */
```

```
            multiple_move_rogue(ch);
            break;
        case 'e':
            eat();
            break;
        case 'q':
            quaff();
            break;
        case 'r':
            read_scroll();
            break;
        case 'm':
            move_onto();
            break;
            case 'd':
            drop();
            break;
        case 'P':
            put_on_ring();
            break;
        case 'R':
            remove_ring();
            break;
#ifndef WONDERSWAN
        case CTRL('P'):
            remessage();
            break;
#endif /* WONDERSWAN */
        case CTRL('W'):
            wizardize();
            break;
        case '>':
            if (drop_check()) {
                return 0;
            }
            break;
        case '<':
            if (check_up()) {
                return 0;
            }
            break;
#ifndef WONDERSWAN
        case ')':
```

```
        case ']':
            inv_armor_weapon(ch == ')');
            break;
        case '=':
            inv_rings();
            break;
#else
        case ')':
            inv_armor_weapon(1);
            inv_armor_weapon(0);
            inv_rings();
            break;
#endif /* WONDERSWAN */
        case '^':
            id_trap();
            break;
            case 'I':
            single_inv(0);
            break;
        case 'T':
            take_off();
            break;
        case 'W':
            wear();
            break;
        case 'w':
            wield();
            break;
#ifndef WONDERSWAN
        case 'c':
            call_it();
            break;
#endif /*WONDERSWAN*/
        case 'z':
            zapp();
            break;
        case 't':
            throw();
            break;
        case 'v':
#ifdef WONDERSWAN
message("Rogue-clone: Version II. ", 0);
message("Tim Stoehr was here,tektronix!zeus!tims", 0);
```

```
message("Japanese edition: Ver.1.3", 0);
message("enhanced by ohta@src.ricoh.co.jp", 0);
message("WonderWitch edition: Ver.0.4", 0);
message("ports by gohodoji@neco.nu", 0);
#else
message("Rogue-clone: Version II. (Tim Stoehr was here), tektronix!zeus!tims ", 0);
#ifndef ORIGINAL
message("Japanese edition: Ver.1.3 (enhanced by ohta@src.ricoh.co.jp)", 0);
#endif
#endif
            break;
        case 'Q':
            if (quit(0)) {
                return 1;
            }
            break;
#ifndef WONDERSWAN
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
            move(rogue.row, rogue.col);
            refresh();
            do {
                if (count < 100) {
                    count = (10 * count) + (ch - '0');
                }
                ch = rgetchar();
            } while (is_digit(ch));
            if (ch != CANCEL) {
                goto CH;
            }
            break;
        case ' ':
            break;
        case CTRL('I'):
            if (wizard) {
```

```
                inventory(&level_objects, ALL_OBJECTS);
            } else {
                message(unknown_command, 0);
            }
            break;
        case CTRL('S'):
            if (wizard) {
                draw_magic_map();
            } else {
                message(unknown_command, 0);
            }
            break;
        case CTRL('T'):
            if (wizard) {
                show_traps();
            } else {
                message(unknown_command, 0);
            }
            break;
        case CTRL('O'):
            if (wizard) {
                show_objects();
            } else {
                message(unknown_command, 0);
            }
            break;
        case CTRL('A'):
            show_average_hp();
            break;
#ifndef ORIGINAL
        case CTRL('G'):
#endif
        case CTRL('C'):
            if (wizard) {
                new_object_for_wizard();
            } else {
                message(unknown_command, 0);
            }
            break;
        case CTRL('M'):
            if (wizard) {
                show_monsters();
            } else {
```

```
                message(unknown_command, 0);
            }
            break;
#endif /* WONDERSWAN */
        case 'S':
            save_game();
            break;
        case ',':
            kick_into_pack();
            break;
#ifndef WONDERSWAN
#ifndef ORIGINAL
        case CTRL('X'):
            if (!wizard) {
                message(unknown_command, 0);
                break;
            }
            draw_magic_map();
            show_monsters();
            show_objects();
            show_traps();
            break;
        case '?':
            help();
            break;
        case '@':
            print_stats(STAT_ALL);
            break;
        case 'D':
            discovered();
            break;
        case '/':
            identify();
            break;
        case 'o':
            options();
            break;
        case '!':
            doshell();
            break;
#endif /* ORIGINAL */
#endif /* WONDERSWAN */
        case 'a':
```

```
            ch = cmd = oldcmd;
            goto CH;
        case ' ':
        case '\033':
            break;
        default:
            message(unknown_command, 0);
            break;
        }
    }
}
```

裏葉:プレイヤーがどのキーを押したらどういった処理をするかが書かれてございますね。
柳也:あと、この関数にも for(;;){} で囲まれた部分があるのに気付いただろう。
神奈:どういうことだ?
柳也:つまり、特に指示がない限りプログラムはこの for(;;){} で囲まれた部分の処理を繰り返すんだ。
神奈:指示とは例えばどういうものなのだ?
裏葉:例えば階段を上ったり降りたりした時でございますね。
柳也:そうだ。その時は clear_level(); の部分に戻らないといけないだろう。
神奈:そういえばそうであるな。
柳也:最後に play_level() から return した時 clear_level(); の部分に戻る前に free_stuff(); を 2 回呼び出だすんだが、まあこれは新しくマップを作る前に必要なデータを保存しているんだと思ってくれればいい。

整理

柳也:じゃあ、最後に rogue が主にどういった動きをしているかだけ大雑把にまとめておこう。

マップを初期化する
↓
新しいマップを作る
↓
アイテムを配置する
↓
階段を配置する
↓
罠を配置する
↓
モンスターを配置する
↓

キャラクターを配置する
↓
ステータス等を表示する
↓
キー入力に対応した処理を行う
(指示があるまでこの部分の処理を繰り返す)
↓
指示があれば情報を保存して最初に戻る。

柳也:まあ、こんなところだな。
神奈:そうか、ごくろうであった。おかげでろーぐがどうなっているのか少しは理解できたぞ。
柳也:ほんとか?
裏葉:それではそろそろ夕餉の用意をいたしましょう。
神奈:そうか、ではまいろうかの。
柳也:そうだな…

往人:…っていう夢を見たって?
観鈴:うんうん。
往人:なんだそりゃ…
観鈴:にはは…

後書

往人:で、なんで俺が後書にまで登場しないといけないんだ?
米田:だって一人で喋るのって疲れるから。
往人:だいたいなんでこんな馬鹿な事しようとか思ったんだ?
米田:こういった話って本とか読んでたら皆堅苦しくって読んでて肩がこるだろ?
米田:で、こういった感じにしてみれば少しは気楽に読めるかな?とか思ったんだけど…
往人:懲りたんだろ。
米田:まあな、書いてるうちに自分がいかに文章力がないかがよくわかったよ。だからもうここでは二度とやらないと思う。
往人:懸命だな。
米田:後な、こういった書き方にすると文章の量の割に内容が薄くなるんだ。今回も量の割にたいした事書けてないし。まあ面白く書こうと思ったら仕方のないことなのかもしれないけど…
往人:結局、内容を濃くしようと思ったら、あの堅い感じの文章になるって事か…
米田:まあな、それに登場人物が誰かわからない人は読んでてもあまり面白くないだろうし。まあ、登場人物がわかってもあまり面白くないという話もあるけど…
往人:じゃあこんなのただの自己満足じゃないのか?

米田:まあそういうことになるな。
往人:まったく…。
米田:まあこれ以上言っても泣き言にしかならないからもうやめるよ。というか今までのでもたいがい泣き言だけど。
往人:そうだな。まあ来年はもう少しがんばれよ。
米田:ああ、そうするよ。

2000 年 11 月 21 日　BGM　縁

実況パワフルプロ野球７　　サクセスモードについて

電子情報工学科　１回生　　岸田　匡司

１．初めに

今回、実況パワフルプロ野球のサクセスモードについて書きますが、私自身はいまだオールAの選手を作ったことがありません。だから、この記事は攻略というよりも個人的な意見としてみてもらえればいいかと思います。またある程度パワプロを知っている人を対象に書いています。

２．投手・野手共通事項

まず投手・野手に共通して言えることは、試合に出場して活躍し、そして試合に勝つことです。試合に出ることができなければ、ほとんど経験点はもらえないし、試合に出ていても活躍する（野手ではとにかく出塁する、投手では長い回を投げ、なおかつ出塁させない）ことができなければ、これまた経験点はほとんどもらえません。そして試合に出て自分が大活躍しても試合に負けると、もらえる経験点は半分になってしまいます。といっても一軍のレギュラーになるまで、試合の勝ち負けはコンピュータの操作するほかの選手が活躍するかどうかにかかっているので運任せになってしまいますが・・・。しかし試合で活躍してチームが勝ったときにもらえる経験点はひじょうに多いので、いい選手をつくろうと思ったら全試合勝つことが一つの目標になります。試合に勝つためには、自分が選んだチームの弱いポジションを自分が作っている選手でうめる、というのがひとつの方法だと思います。

次に大切なことはチームごとに特色があり、もらえる経験点や取得しやすい特殊能力に差があるので、どんな選手を作りたいかによって、その選手にあった適切なチームでゲームをはじめることです。たとえば、足の速い選手を作りたいと思うならば西武ライオンズや中日ドラゴンズといったふうにです。これはかなり重要でたとえば、走塁系のレベル１の練習のダッシュを中日ドラゴンズと読売ジャイアンツで行うと、いちどにもらえる経験点には４ポイントもの差がつきます。

この記事では次章以降、上の２つのポイントを中心に、投手編・野手編、さらにチームごとに分けて解説をしていきます。

３．投手編

選んだチームに関係なく、投手全体にいえることを挙げたいと思います。まず最初にいえることは、スタミナはある程度はぜったいに必要だということです。これは抑えタイプのピッチャーを作るときにもあてはまることで、監督の評価が低いときは中継ぎとして２，３回投げるときもありますが、基本的にはかならず先発として使われるからです。だからスタミナがないと、途中でスタミナ切れで交代させられたり、球速が落ちた

り変化球のキレがなくなって打たれたり、とあまりいいことはありません。そういうわけでスタミナは少なくともDに近いEぐらいは欲しいところです。次に、CPUが操作しているときは、変化球の種類が多いほど試合で打たれにくい、ということです。ただし変化球の種類を多くすると、一つ一つの変化球の変化量を大きくすることが難しくなります。

４．野手編

野手をつくるときに大事なことは、どの能力から上げていくかということです。上でも書いたとおり試合で活躍することが大事なのですが、試合の評価は打撃だけで決まり、守備にはまったく関係ありません。だからパワーやミートカーソルをはじめに上げてしまうのがいい方法です。その次に優先したいのは走力です。試合では一塁打よりも二塁打、二塁打よりも三塁打のほうが経験点が多くもらえるからです。しかし個人的に走力は上げにくいと思うので、それほど無理をして上げる必要も無いと思いますが。次にポジション選びも重要です。育成選手がスタメンになると、それまでスタメンだった選手が試合に出られなくなるので、ポジションが悪いと主力選手を欠くことになってしまいます。また守備力が低いと自分が選んだポジションに関係なくファースト（またはパ・リーグならDH）を守ることがあります。

５．チーム別解説

５－１．中日ドラゴンズ
上げやすい能力・・・ミートカーソル、走力、守備力
上げにくい能力・・・変化球
狙いやすい特殊能力・・・逃げ球、ムラッ気

中日で選手を作るとしたら投手にかぎると思います。中日で投手を作る場合の最大のメリットは、なんといっても１２球団で唯一オリジナル変化球をおぼえられるということです。これはオリジナルキャラクターでチームメイトの阿畑と遊ぶことによって発生するランダムイベントによって取得できる。ちなみにオリジナル変化球をおぼえるイベントで断ると逃げ球を取得できます（ただしこの場合、オリジナル変化球はおぼえられない）。野手を作るなら守備力の低い選手がスタメンのショート（福留）かサード（ゴメス）がいいでしょう。

５－２．読売ジャイアンツ
上げやすい能力・・・球速、スタミナ、パワー
上げにくい能力・・・走力
狙いやすい特殊能力・・・尻上がり、安定感、広角打法

ジャイアンツではカーブの投げられる投手を作るのがおすすめです。なぜならランダ

ムでＯＢが出現して「怪物カーブ」という特訓を教えてもらえることがあるからです。この「怪物カーブ」という特訓は１０％の確率でカーブの変化球レベルを＋１してくれるという、すばらしいものです。また野手を作るのなら、唯一のウィークポイントといってもよい捕手か、松井・高橋以外にバランスのよい選手のいない外野手がいいでしょう。ちなみにオリジナルキャラクターの猪狩は、投手でありながらパワーAかつパワーヒッターの特殊能力を持っているので、かなり強力な打線になっています。さらに猪狩が先発なので、上原を抑えに使うことができる。

５－３．横浜ベイスターズ
上げやすい能力　／　上げにくい能力・・・なし
狙いやすい特殊能力・・・威圧感、打球反応○

横浜では投手の場合、ひじょうにレアな特殊能力である威圧感を取得できる。これはランダムイベント（オリジナルキャラクターの伊達の本を読む）で、しかも取得できる可能性は６％と低いので運に任せるしかないが、その効果は絶大です。それに加え、OBが「魔人フォーク」という特訓（１０％の確率でフォークの変化を一段階上げる）を教えてくれることもあり、味方打線の強力さとあいまって、投手を作りやすい環境だといえます。野手に関してはあまりメリットがありませんが、作るのなら一塁手か外野手がおすすめです。

５－４．ヤクルトスワローズ
上げやすい能力・・・球速、守備力、ミートカーソル
上げにくい能力・・・コントロール、パワー
狙いやすい特殊能力・・・低め○、リリース○、勝ち運、尻上がり、サヨナラ男、
送球○、代打男、ムードメーカー

ヤクルトでは取得できる特殊能力が多いが、すべてとることは不可能です。なぜなら、ほとんどの能力が同じイベント関連（マラソンとオリジナルキャラクターの日下部からもらう本）なので、一つとるとほかのものがとれなくなるからです。ちなみにサヨナラ男はヤクルト以外では取得することができません。育成するのにおすすめのポジションは古田とペタジーニのいる捕手と一塁手以外です。ほかのポジションでは可もなく不可もなくといった選手しかいません。投手の石井（一）もコントロールが極端に悪く、そこがネックになっています。

５－５．広島東洋カープ
上げやすい能力・・・ミートカーソル、走力、守備力
上げにくい能力・・・なし
狙いやすい特殊能力・・・重い球、キレ○、リリース○、打たれ強い、威圧感、
連打○、逆境○、ヘッドスライディング、固め打ち
広角打法

上で挙げた特殊能力はすべてオリジナルキャラクターの嵐暴コーチがらみです。ただし強制的に練習させられたりするイベントがあるので、広島で特殊能力を狙うと、うまくいった場合はすばらしい選手ができると思いますが、失敗するとケガばかりでまともに選手を作ることもままなりません。かなりハイリスクハイリターンな球団だといえるでしょう。広島では外野手にはいい選手（緒方、前田、金本）が多いのでそのほかのポジションがいい。また野手を育成する場合、頼れる投手が佐々岡一人なので、試合で延長になるとかなり苦しくなる。

５－６．阪神タイガース
上げやすい能力　／　上げにくい能力・・・なし
狙いやすい特殊能力・・・勝ち運、ポーカーフェイス、逆境○

阪神はこれといった特徴もないので、安定して選手を作ることができる球団だと思います。作るとすれば内野が弱いので（特に二・三塁）、ここを育成選手で埋めるのがいいのではないでしょうか。ただし、オリジナルキャラクターの矢沢に関するイベントが発生すると、特訓をかってに変えられてしまう（瞑想や盗みになる）ことが多いような気がするので、そこだけは注意が必要です。

５－７．福岡ダイエーホークス
上げやすい能力・・・ミートカーソル、パワー
上げにくい能力・・・球速、スタミナ、走力
狙いやすい特殊能力・・・ムード○、対左打者○、対左投手○、体当り、
チャンス○

ダイエーでは先発ピッチャーと二塁手の能力が低いので、そのポジションで育成するのがいいでしょう。投手を育成するときには、球速とスタミナが上がりにくいという欠点がありますが、打線が強力なので少しぐらい点をとられても大丈夫という、利点もあります。サクセスモードではなぜかスタメンキャッチャーが坊西になっているので、すぐに城島に交代させたほうがいいです。またオリジナルキャラクターのボムを追いかけるイベントでは、「左へ」を選ぶと投手なら対左打者○を、野手なら対左投手○をそれぞれ１００％取得することができます。

５－８．西武ライオンズ
上げやすい能力・・・球速、走力
上げにくい能力・・・パワー
狙いやすい特殊能力・・・回復○、キレ○、ノビ○、ムード○

西武で作る場合、投手・野手に関わらずオリジナルキャラクターの一文字とのイベントはあまりうれしいものがない。特に投手のときに発生するフォームチェンジイベント

は成功しても能力が下がるので、発生したら断ってイベントを終了させたほうがいいと思います。また、投手にかぎりOBが登場して「神様スライダー」という特訓を教えてくれることがあります。これは１０％の確率でスライダーの変化を一段階上げてくれるので、西武で投手を作る場合スライダーを覚えさせるのがいいでしょう。野手では、サードとセカンドがいいと思います。

５－９．オリックスブルーウェーブ

上げやすい能力　／　上げにくい能力・・・なし
狙いやすい特殊能力・・・安定感、クイック○、尻上がり、流し打ち、内野安打○、
キャッチャー◎、アベレージヒッター

オリックスで作るときは、外野にはすばらしい選手（イチロー、田口、谷）がはじめからいるので、内野（三塁手）や、頼れる投手がオリジナルキャラクターの神童ぐらいなので投手を選ぶのがいいと思います。特にオリックスにはパワーが高い選手がいないので、パワーAの内野手を目指して育成すれば試合が楽になるでしょう。また他のいくつかの球団のようにOBが「サブマリンシンカー（シンカーの変化が１０％の確率で一段階上がる）」という特訓を教えてくれることがあるので、シンカーを決め球に持つ投手を作るのに適しています。

５－１０．千葉ロッテマリーンズ

上げやすい能力　／　上げにくい能力・・・なし
狙いやすい特殊能力・・・回復○、ムード○、リリース○、粘り男、初球○、
キャッチャー○

ロッテの野手では遊撃手の小坂以外、とびぬけて優れた選手はいないので、どのポジションでつくっても問題はないでしょう。悪い見方をすればなかなか試合では勝ちにくいチームだと思います。また投手の場合、オリジナルキャラクターである早川からマリンボール（Hシンカー）を教えてもらえる。ただ、ある程度早川の評価が高くないといけないのですが、早川の評価は下がりやすいのでHシンカーを狙うときには注意が必要です。

５－１１．日本ハムファイターズ

上げやすい能力・・・パワー
上げにくい能力・・・球速、スタミナ
狙いやすい特殊能力・・・牽制○、リリース○、尻上がり、ポーカーフェイス、
固め打ち、粘り男、ヘッドスライディング、初球○、
流し打ち

日本ハムはなんといっても投手力が低いので投手を育成すれば、試合に勝ちやすくなるでしょう。しかし球速は上げにくいので、軟投派ピッチャーがいいと思います。野手

を作るなら走攻守そろった選手のいない外野手を作るのがいいと思います。オリジナルキャラクターはチェリーですが、本来のセカンド金子のほうが能力的には上なので、交代させたほうがよい。ちなみにチェリーと競うことになるカード集めでは、OBのカードが出るとその選手をほかのモードで使うことができるようになります。

５－１２．大阪近鉄バッファローズ
　上げやすい能力　／　上げにくい能力・・・なし
　狙いやすい特殊能力・・・回復○、いいひと、ケガしにくい、固め打ち、守備職人

　大阪近鉄でも投手力の弱さが目立っています。さらに中継ぎエースの香田が登場しないので、かなり戦力ダウンしています。よって投手を育成するのがいいと思います。野手では二塁手がいいでしょう。オリジナルキャラクターの今宮が困っているときは、お金を渡すこと。そうすれば回復○などのよい見返りが必ずあります。またもう一人のオリジナルキャラクターである内野から守備職人を取得できるイベントがありますが、取得できる確率は自分の守備力に比例します（守備力が最高の１５なら６０％）。

JNETHACKをやってみよう

機械システム工学科　1回生　鹿田　暢亮

ＮｅｔＨａｃｋは、ＶＡＸという、ＵＮＩＸマシンに付属していたコンピューターゲームです。内容は、ダンジョンに潜り、Ｙｅｎｄｏｒの魔除けなるものをとってくるというもので、トルネコの冒険や風来のシレンの元ネタのゲームなのです。かなり古くからあるゲームですが、その面白さはトルネコ等以上のものです。ここで、ここでＮｅｔｈａｃｋの日本語版であるＪＮｅｔｈａｃｋを紹介したいと思います。

Ｎｅｔｈａｃｋはまあ、基本的にトルネコとおなじです。まず、１３の職業の中からひとつを選んではじめます。プレイするたびにダンジョンが変化し、モンスターを倒しながら階段を降りて下に潜っていきます。モンスターに殺されたり、罠にかかったりおなかが減ったりして死んでしまったらゲームオーバーです。

Ｎｅｔｈａｃｋをプレイしてまず驚くのは、画面が文字キャラだけであるということです。どういうことかというと、主人公つまりあなたの操作するキャラは@です。通路は＃だし。オークが襲いかかってくるときは、Ｏ（オー）が近寄ってきて殴ってきます。これだけみると絵がついているトルネコやシレンのほうが面白いと思うかもしれませんが、それ以上に、Ｎｅｔｈａｃｋには面白い要素があふれています。

１．多くのコマンド

Ｎｅｔｈａｃｋにはとても多くのコマンドが存在します。例えば、神に祈る、供物を捧げる、顔を拭う、アンデッドを土に返す、何かに物を浸す、物を蹴る、床に字を書くなどなどでこれらのコマンドでいろいろなことが出来ます。『緑色の薬』を『何かを物に浸す』コマンドで泉に浸すと『薄まった緑色の薬』になりさらにもう一回やると『無色の薬』になってしまいます。神に祈ると神の機嫌が良いときはＨＰが回復したりお腹がふくれたりといいことがおきます。逆にご機嫌斜めだとレベルを下げられたりします。で、はじめは機嫌の良い神様ですが祈りまくっているとご機嫌斜めになってしまいます。そういうときは神に供物を捧げるわけです。自分の神様の祭壇を見つけてモンスターの死体を持っていって『供物を捧げる』コマンドを使うと死体が燃え上がり神さまの機嫌が良くなります。『物を飲む』コマンドを泉の上で実行すると泉の水を飲むことが出来ます。ここでよくいろいろな事が起こります。泉から蛇がわらわらわいてきて噛まれまくったり、宝石を見つけたり、ジンが出て来たりもします。しかもこのジンはめちゃめちゃ強くて殴られたらすぐ死にます。けどこのジン、機嫌が良いと『１つだけ何でも好きな物をやろう。何がお望み？』といってアイテムをなんでも１つくれて消えていくとってもいいヤツです。こんな風にＮｅｔｈａｃｋにはそこらのゲームとは比べ物にならないくらいのギミックがあってこれがＮｅｔｈａｃｋを面白くしています。はじめ、プレイするうちはとっつきにくいですけど慣れればかなり面白いと思います。

２．いろいろな職業

Ｎｅｔｈａｃｋには１３の職業が存在します。考古学者、騎士、戦士、ワルキューレ、野蛮人、洞窟人、薬師、僧侶、魔法使い、エルフ、観光客、盗賊、侍です。
これらの職業はそれぞれの個性的な能力と装備を持ってダンジョンに入っていきます。ここで、いくつかの面白い職業について少し書いてみます。まず観光客です。ダンジ

ョンでモンスターと戦うゲームの職業にもかかわらず、なぜか観光客。ご想像のとうり、めちゃめちゃ貧弱です。初期装備がなんとアロハシャツそして大量の食い物とお金、そしてカメラなどなどというなんともヤル気のない職業です。ＨＰや攻撃力もやたら低くて、モンスターと初めからまともに戦うのも難しいです。で、どうやって戦うのかというと、ヤバくなったらカメラでパシャッとやってモンスターがびびっている間にすたこら逃げ出すのです。さらに貧弱極りないことは、鍵がかかっているドアを無理やり蹴って開けようとすると、たまに失敗してなんと死んでしまうことがあります。このように観光客はプレイが非常に難しく、まったく初心者には向いていない職業です。

それに対して初心者にも使いやすくてなかなか強い職業が、野蛮人とワルキューレです。こいつらは、初期ステータスがかなり強くて、適当に殴っていっても結構奥まで進めます。さらにワルキューレは初めからかなり硬い盾を持っていて防御力が高いです。

Ｎｅｔｈａｃｋではよく食料が無くて困ることがあります。しかし、このゲームではモンスターの死体を食べて飢えをしのぐことができます。ですがモンスターを食べるとよく食毒を起こすことがあります。食毒を起こすとほとんど即死と変わりません。数歩いただけで一発昇天です。しかし野蛮人は普段から悪いものを食っているせいか、毒への耐性があって、腐ったものを食おうと余裕で戦えます。ワルキューレは初めからそこそこの食料をもっているし、装備があまり重くないので結構食いつなぐことができます。

次に、戦士ですがこいつは普通の戦士とは一味違います。なぜか必ず性別が女。ペットの名前はアルテミス。そして初期装備の鎧を見てみると、『セーラー服』 えっ？セーラー服？そして画面の下をみてみると、 『ｓｈｉｋａｔａ マーキュリー』と書いてある。ええ～っ。そう、こいつは某セーラー○ーンに登場する某セーラー○ーキュ○ーなのです。なぜマーキュ○ーなのかは謎で、たぶん作ったやつの趣味でしょう。

この職業は初めから変化の杖なるものを持っています。これはなかなか面白いアイテムで、ためしに横にいるアルテミスに使ってみると、運がよければ、貧弱な猫が力強いドラゴンに変わったりします。

他の職業もいろいろと特徴があって面白いです。皆さん試してみてください。

３．最後に

ｊＮｅｔｈａｃｋはｈｔｔｐ：//www.ｊｎｅｔｈａｃｋ.ｏｒｇでダウンロードできます。やりたいと思った人はさっそくダウンロードしましょう。ここには攻略などものっていますから見てみるといいですね。

こんな紹介を書いている私ですが、あまりこのゲームをやりこんでません。この文章自体もかなりテキトーです。まあ、これを読んでＮｅｔｈａｃｋをやってみて面白いと思ったひとがいればうれしいです。

CDについての簡単な内容

機械システム工学科　一回生　清水俊伸

@論理フォーマット

CD-Rが普通のCD-ROMドライブなどで再生できるのは、CD-Rの論理フォーマットが音楽CDやCD-ROMと互換性を持っているためである

ＩＳＯ9660

国際標準フォーマット。MS-DOSからはレベル1のCDは読むことが出来るがレベル2、3のCDは読めない

Rock Ridge

ISO9660をもとにUNIX環境に適合させたもの

Joliet

ISO9660の拡張で、IDO9660レベル1で8文字　拡張子3文字まで、レベル2、３でも31文字までなのに対して、64文字までのファイル名をつけることが出来る　Windows95のために開発された

Romeo

ISO9660の拡張で、128文字までの長いファイル名を扱えるが、Windows95/98/2000,NT4.0でしか読み出せない　Windows NTのために開発された

UDF

Universal Disc Format　パケットライトで使うためのものだが、元はDVD-ROMの規格だった

HFS

Macintosh用

@いろいろなCD

ＣＤ-ＲＯＭ

Compact Disc-Read Only Memory 読み出し専用

Mode1はデータ向けでMode2は動画向けになっているが、混在は不可

Video-CDやCD-Iは２の方で書き込まれている

CD-R

パソコン用のCD-R/RWドライブで使用できる汎用の製品と音楽専用CDレコーダー専用の二種類がある

データの記録が可能だが、書き換えは出来ない

製造メーカーは太陽誘電かTDKがいいようである

保護層　反射層　記憶層（有色素）　基盤　から成っている

CD－Rの記録方法

（1）ディスクアットワンス

Non Interrupt Recordingとも言われ、信号をリードイン、データ、リードアウトの順に１回で記録していく方法　そのためデータは、一筆書き状に配置される　追記は出来ない

（2）インクリメンタルライト

複数回の書きこみが可能

「1」　トラックアットワンス

１回のデータ書きこみを１トラックとして、追記していく方法

「2」セッションアットワンス

リードイン、データ、リードアウトを１セッションとし、セッション単位で追記する

書きこみ方法

「3」パケットライト(Packet Write)

パケット単位で追記する方法　パケットライトソフトが必要

（3）マルチセッション

1つのセッションはリードイン、データ、リードアウトにより構成されていて、このセッションが1枚のディスク上に複数存在するもの

CD-RW

書き換え可能ではあるが、書き直すときは、一度データを居一括消去する必要がある

CD-I

CD-DA、オーディオ情報、画像情報、データや文字情報などを記録できる

CD-ROM-XA

CD-ROM extended Architecture 長時間の音声データや大量の画像データが扱える　CD-ROMにCD-Iの技術を取り込んだもの

Fourm1はデータ向けで、fourm2動画向けに記憶容量を重視、となっているが混在が可能

CD-DA(音楽CDの標準フォーマット)

CD Digital Audio　最大99トラックまで格納でき、一般的に1トラックあたり1曲　CD-ROMとは物理仕様は同じで、円盤の中心孔から追加CD-R領域、リードイン、プログラム領域、リードアウト、.となっている　44.1kHz

基盤のポリカーボネート樹脂にアルミニウムを蒸着し、保護層でカバーした構造

シングルCD

CD-DAのシングル版

シングルCD-R

シングルCDと同じサイズのCD-R

CD-Extra

音楽用CDに、パソコン用のマルチメディアデータを追加したもので、CDの最初のセクションに1つ以上のオーディオトラックを、第2にセクションに1つのCD-ROM XAデータトラックを含むマルチセッションのCD

Enhanced CDと呼ぶこともあるが、厳密には異なり、さらに類似の規格にCD Plusもあり、区別は大変難しい

Mixed Mode CD

データトラックの次にオーディオトラックのあるもの

CD-G(Graphics)

静止画のグラフィックスデータを含む音楽CD

一部のカラオケタイトルなどが採用しているが、画像データはサブコードの未使用エリアに記録するので情報量が少ない静止画を数枚記録するのが限度だが、７４分の再生が可能

CD-EG

グラフィック情報が8色のCD-Gを256色に拡張したもの

CD＋MIDI

CD-DAにMIDI(Musical Instrument Digital Interface)と呼ばれる電子楽器の演奏情報を入れたもの

CDV

20分のCD-DAに加えてディスクの外周部に5分のアナログ動画（デジタル音声付）が記録できる

CD TEXT

5000文字までのCD情報をセッションの先頭部分のリードイン内にあるTOCに記録したも

の
HDCD
音楽CDの規格の範囲内で高音質化した音楽CDで構造も同一
HDCD
音楽CDの規格の範囲内で高音質化した音楽CDである　HDCD対応プレイヤーでは20ビット相当の再生になる
DTS CD
マルチチャンネル録音の音楽CDで、DTS対応のプレイヤーが必要
RIAA CD
レコードの録音手法を応用し、低音域のレベレを下げて、高音域のレベルを上げている
マルチトラックCD
マルチトラック録音したデータをCD-Rに書きこんだもの　CDプレイヤーでは再生できない
Dolby Digital CD
Dolby AC-3のデータをCD-Rに書きこんだもの　パソコンで再生してAVアンプと5.1チャンネルスピーカで楽しめる
XRCD
K2プロセッシング技術を使って製作過程を20ビット化している
高温質マスターテープ使用の音楽CD
可能な限り高音質化した音楽CD
FMV(Full Motion Video)
CD-Iの規格にデジタル動画を加えたもの
Video-CD
MPEG1形式のビデオデータを記録できる
Photo-CD
最大99枚まで写真データを記録可能
ﾌﾞｰﾀﾌﾞﾙCD
起動システムが含まれていて、パソコンを起動させることが出来る

@その他
EP-WING
電子化することでメリットの大きい辞書などをCDに格納するための規格で、この規格のCDはISO9660に準拠しているためパソコンなどでも利用できる
地図用CD-ROM
カーナビゲーション用に地図データを記録したもので、メーカーによっては独自の形式のCDを使用している場合もある
DVD-Video
タイトルは音楽系もあるが映画系がメインである
スーパーオーディオCD(通称SACD)
DSD方式の録音で100KHzに近い再生帯域を144デシベルのダイナミックレンジで再現できる
DVD Audio
PCM方式の録音で基本的にマルチチャンネル再生　再生帯域は96KHz

DVD-RAM，DVD-Rなどもある

工芸的プログラミング～壱～

工芸学部電子情報工学科　越本　浩央

平成12年11月21日

時は世紀末
電子計算機の誕生から約半世紀
ある者は勝ち、ある者は敗れ …
敗れ去った勇者の伝説を、
私達が語り継ぎましょう …

1　近頃のプログラミング

正直に言いまして、私は若者です。ええ、ほんと年長者には敵いません。けれども、言いたい事はあります。そりゃもう沢山。そんなわけで、まずは近頃のプログラミングについてちょいとヤカラを …

今時誰だってOOPを知ってます。それこそ中坊もとい厨房だって。なんたって流行りはJAVAです。ほんとあれって便利ですよね。何から何までライブラリが揃ってるし、"Write once, Run anywhere"なんですよ。従来のプログラムに比べて重いのがアレですが、いいんです、だって近頃のプロセッサーは高速で高機能なんですから。でも、やっぱり応用アプリケーションはC++です。Cはもう使われてませんよね。システムウェアはまだCなのに。何故か？やっぱりOOPだからですか？自分のバグを誤魔化せないからですか？

そう言えば、Bjarne Stroustrupの「プログラミング言語C++（第二版）」高いですねぇ。いや別に中身が無いわけじゃないんですよ。立ち読み程度での様子では中身はしっかりしてる。まぁC++の作者ご本人が書いてるんだから当然でしょうけど。それでも絶対に高いですよ。だってあんなに勉強して、見返りはC++が使えるだけなのに …

MacOSXが出ますね。GUIのAquaってとても魅力的。インターフェースだけじゃなくて、グラフィックＡＰＩも魅力的。メタファーの追求をやめた（今回から窓の操作に近づいた）のは少しがっかりですが、それでもJovesが帰ってきたことでなんだか興味が湧いて来ます。それにも増して、Machの血筋になることは素敵ですよね。特に、NeXTの後継的なところが。いや、つまりObjective-Cのフレームワークであるということがね … それなのに、林檎系列の雑誌ではクラシックなことばかり。古いソフトの動作に執着するようなレヴェルなら、さっさと環境の整った窓にでも移行すれば良いのに …

ここ最近のことを振り返ってみると、こんな感じですか。特撮のケテルビーって曲で、"猫かと思えばよく見りゃパン"という歌詞がありましてけど、ほんとそんな気がしますねぇ。だって …

2　OOPって食べられるんですか？

先の項で述べたとおり、近頃はC++なようです。みんな、"オブジェクト指向でプログラミング（以下OOP）だ"なんて言ってます。クラスでがっちんがっちんのぼこぼこのライブラリ書いて、それはもう洗練されたプログラミング環境を提供してくれますよね。MFCなんかそりゃもう …って、否！断じ

て否 !!!

C++信者など地獄へ落ちてしまえぇ!思想の乏しい愚か者などぉ、地獄で悔いるがEぃー!はぁはぁはぁ…

で、やっぱOOPは新しいパラダイムなんですよ。これははっきりと言えます。そしてこの方向性を持つことは、決して間違いではない。OSがオブジェクト指向を取り入れて設計されるのは当然の成り行きです。けれども、けれども、C++はOOPじゃーない。断じてない。

じゃあ何がOOPか、というのはもっともなわけで、実際ほとんどの人が形式的に定義を言えるわけじゃない。第一、形式的な定義がOOPを説明してくれるかというと、それは別問題。どうせパラダイムなんだから、体で憶えるべしです。そんなわけで、Smalltalkを薦めるかと言うと、それも違うんだなぁ。ましてや、Eiffelなんかでもないんですねぇ。Eiffelも良さ気ですけどね。今回取り上げるのはObjective-C。あのNeXTSTEPが主要言語として採用したオブジェクト指向言語です。

3 Objective-C入門

Objective-Cは、Stepstone社を創立したBradCoxによって開発されたもので、もともとはSmalltalk+Cをモチーフに設計された言語+コンパイラ+ライブラリでした。つまり、プログラミング環境だったわけです。ところが、開発元のStepstone社はこれを広めることが出来なかった。1985年、Apple社を去ったSteve Jovesが、m68k機とNeXTSTEPオペレーティングシステムの開発を行うNeXT社を創立し、そのマシンのインターフェースをDisplay PostScriptとObjective-Cで書いたことが、Objective-Cの知名度を挙げました。実際、その開発力が見とめられ、NeXTコンピュータの主力言語となりました。特に、この言語の優れていたとろこは、Smalltalkばりの純粋なOOPを取り入れつつ、Cの上位互換であることの利便性、そして強力な動的実行環境にあります。

ここで簡単にObjective-Cの魅力を列挙すると、

- 動的型決定
- 動的クラス拡張
- スマートな文法
- シンプルなクラス構築
- 実行時メソッド決定
- ANSI-Cの上位互換
- (将来的に) Write once, Run anywhere

という感じです。まぁもちろん欠点もあって、

- Cの欠点はやはり残る
- 本質的な多重継承がいない(将来的にも)
- クラス変数が無い
- メソッドは全てpublic(強制的な保護が無い)
- ガーベッジコレクタは無い
- オーバーヘッドが結構ある

となります。**しかし、しかぁしっ!これは新しいパラダイムなのだから、ただ単に速度の問題や、これまで普通に使われたクラス変数のことで、その言語の価値を判断するのは間違いなのだ!**

では、早速その一端でも皆さんに知ってもらうとしましょう。

まず、Objective-Cではクラスは全てObjectクラスからの単一継承で構築されます。各Objectの持つメソッドの実行は、クラスのインスタンス(実体。変数と思いねぇ)にメッセージを送ることで行われます。基本はただそれだけ。簡単でしょう。もしかするとJAVAより楽ではないでしょうか?もちろん、Cを知っているという前提がありますが…

次に、クラスのインスタンスを生成する為に必要なのはidと呼ばれる型です。Cのvoid *みたいなものです。生成は、作りたいクラスにnewというメッセージ送ることで終わりです(ここではまだ実体化されていないクラスにメッセージを送ります)。JAVA

が参照によってオブジェクトとの接続を保つのと同様に、Objective-C でも実体に対し、id 型の変数を介して参照します。

オリジナルのクラスの作り方は、Object 型を継承することから始まります。内部変数は、外部からは基本的に一切手が付けられません。全てクラス間のメッセージのやり取りで処理していきます。効率的にクラスを構築する為に category や protocol など、JAVA でいう interface みたいなもの（共通したメンバ変数やメソッドをまとめて宣言）があります。

そして、ここが Objective-C の真骨頂なんですが、class 型を利用して実行時にクラスの型を判別することが出来ます。selector 型を使用して、任意のメソッドがクラスに所属するかどうかを判別出来ます。更に、selector 型を介して任意のメソッドを実行できます。文字列から selector 型を作れます。これだけ出来れば、動的に処理を組みかえることが出来ます。つまり、言語仕様に沿って楽〜に動的なシステムを構築できます。また、クラスには protocol を実行時に選んで添付出来ます。実行時に添付した protocol を判別できます。これを使うと、ネットワークを介して、違う実装のクライアントを判別し、それに応じて処理を適応できます。

どうです？魅力的でしょう？えっ、別に C でも出来るって?そりゃもちろん、コンパイルすれば同じバイナリですから出来るでしょうけど、C の場合は非常に処理系に偏ったプログラミングになるでしょう。特に、オブジェクト形式に非常に依存します。しかし、Objective-C なら言語仕様に沿って実現できます。そして、自ら備える動的処理機構で、クロスプラットフォームを実現します。同じオブジェクト指向言語を名乗りながら、C++とは全く違う環境でしょう。そもそも C++は、今使えることを前提として設計されているので、先進性について比べるべきではないのですが …

4 実践？

さて、それでは早速 Objective-C なプログラムを見てみましょう。稿末にソースを載せてあるのでそちらを参照しながら読んで下さい。今回は、X-Window の基本ライブラリ（Xlib）の簡単なラッパーを製作してみます。

まず、X プログラミングを行う際に面倒なのが、似たようなコードを長々と何度も書くことがあるということです。これは確実に生産力を下げます。よって、OOP を行う上でこの問題は解決せねばなりません。OOP が求められた要因、つまり巨大化するシステムを効率的に管理し、現実世界をコンピュータ内部でシミュレートする、ということを考えれば当然です。

そして、X-Window はイベントドリブンプログラムを要求するので、プログラミングは基本的にイベントに対応した処理と、処理されるデータの管理に専念することとなります。逆に言うと、それらのことだけ行えば、プログラムが仕上がります。

一気に結論に持ちこんでしまうと、X-Window の要求するデータ構造は基本的に管理したくない・イベントと扱うデータだけを自分で処理し、それらを簡単に X に実装させたい・なるたけ（直接いじるのが）無駄なコードは隠したい、ということです。以上の要求を満たすようなラッパーを考えると、なぁんだ、Motif みたいなもんじゃない。

で、早速 xbutton.m というソースをご覧下さい。（あぁ、ちなみに Objective-C のソースは拡張子が m です）XManager というクラスに必要な物を登録し、その登録したウィンドウごとにイベントが追加できます。メッセージの配信は XManager の方でやってくれるので、どんどんウィンドウやイベントを追加しても、ライブラリのユーザからは自分のやる仕事以外のコードが膨らむことはありません。また、X-Window 自体の面倒な処理の面倒を見る必要もありません。非常に簡単（かつ不完全）な例ですが、OOP が必要とされた要因はちゃんと解決しています。また、ウィンドウごとにプロパティを追加してその管理も任せたいときは、ライブラリで提供して

いるクラスを継承して属性と属性の管理メソッドを追加する（is-a 関係）か、ウィンドウのクラスに id 型のメンバ変数を追加し、その変数への参照経路を用意する（has-a 関係）、だけで済みます。

XManager はどういう処理をしているのでしょうか。実は大した事をしていません。実は C++で作りかけていたものを急遽変更したものなので Objective-C プロパーなコードが書けていません。非常に残念なことですが、Objective-C の入門材料には良いのではないでしょうか？:-p

XManager 側の処理を一気に流すとしましょう。初期化処理で、ディスプレイやルートウィンドウの設定を行います。ウィンドウの追加では、単方向リストでウィンドウクラスを増やしていきます（ここはネームと対応したクラスへのポインタのセットをマップとして持っておいて、ハッシュテーブルを用意する方が、後々の為には良いかもしれない）。各ウィンドウはイベントクラスのポインタを持っていて、イベントの追加もウィンドウと同様な方法で行われます。イベントループは、XManager の run で処理されます。この中では、メッセージの配信が行われます。配信はリストの順番にチェックを行い、合致したウィンドウの中で、イベントのタイプごとに再びチェックします。実際のイベント処理は、登録した関数をポインタを介して呼び出します。以上が大まかな流れです。

さて、実際のプログラミングではこのライブラリは役不足です。まず何より、先に挙げたウィンドウごとの属性及び関連データの管理機構がありません。ここから先の拡張は読者の皆さんにお任せするとして、そろそろ本稿も終わりにするとしましょう。

5 光を …

ここまで魅力を散々謳っておきながらアレですが、Objective-C はほんとにマイナーな言語です。資料はほとんどありません。今現在もっとも容易に手に入る、そしてもっとも正式な言語資料は、Apple 社の Web サイトにある「Objective-Oriented Programming and the Objective-C Language」という pdf ファイルです。コンパイラの方は、GCC のバージョン 2.4 以降なら対応してます。また、Objective-C の紡ぎ出す洗練されたインターフェースを持つ OPENSTEP のフリープロジェクトである GNUSTEP があります。興味を持たれた方は是非そちらの方をご覧下さい。

```
// Objective-Xlib.h
//        ver.0.1
// (C)Hiroo Coshimoto
#ifndef _OXLIB_H_
#define _OXLIB_H_

#include <objc/Object.h>
#include <X11/Xlib.h>

@interface OEvent : Object                                                    10
{                                                                             11
        int e_type;                                                           12
        id (*e_proc)(id, XEvent *);                                           13
        id next_event;                                                        14
}                                                                             15
- (void)init_event:fp:(int)type;                                             16
- (void)add_event:fp:(int)type;                                              17
- process:xm:(XEvent *)event;                                                18
@end                                                                          19
                                                                              20
@interface OWindow : Object                                                   21
{                                                                             22
        Window w;                                                             23
        char *w_name;                                                         24
        id next_window;                                                       25
        id event_proc;                                                        26
        int e_count;                                                          27
        long e_mask;                                                          28
}                                                                             29
- (void)set_root:(Window)root;                                               30
- init_window:xm:(const char *)master:(const char *)my_name:(int)start_x:(int)start_y:   31
                (int)size_x:(int)size_y:(int)border:(unsigned long)col1:(unsigned long)col2;   32
- (void)make_window:xm:(const char *)master:(const char *)slave:(int)start_x:(int)start_y:   33
                (int)size_x:(int)size_y:(int)border:(unsigned long)col1:(unsigned long)col2;   34
- (void)add_event:(Display *)dpy:(const char *)w_target:fp:(int)type:(long)mask;   35
- (Window)get_window:(const char *)w_target;                                 36
- run:xm:(XEvent *)event;                                                    37
@end                                                                          38
                                                                              39
@interface XManager : Object                                                  40
{                                                                             41
        Display *dpy;                                                         42
        char *dpy_name;                                                       43
        int screen;                                                           44
        id root;                                                              45
        GC gc;                                                                46
        XEvent event;                                                         47
        void (*usual_proc)(id, XEvent *);                                     48
}                                                                             49
- free;                                                                       50
- (Display *)init_manager:(const char *)my_name;                             51
- (void)add_window:(const char *)master:(const char *)slave:(int)start_x:(int)start_y:   52
                (int)size_x:(int)size_y:(int)border:(unsigned long)col1:(unsigned long)col2;   53
- (void)add_event:(const char *)w_target:fp:(int)type:(long)mask;            54
- (void)add_usual_proc:fp;                                                   55
- (void)map_window:(const char *)w_target;                                   56
- (void)map_windows:(const char *)w_list;                                    57
- run_loop;                                                                  58
- (Display *)get_display;                                                    59
- (Window)get_window:(const char *)w_target;                                 60
- (int)get_screen;                                                           61
```

```
- (GC)get_gc;
@end

#endif
```

```
//Objective-Xlib
//      ver.0.1.
//(C)Hiroo Coshimoto
#include "OXlib.h"

@implementation OEvent

- (void) init_event : fp : (int) type
{
        e_type = type;
        e_proc = (id (*)(id, XEvent *))fp;
        next_event = nil;
}

- (void) add_event : fp : (int) type
{
        if (next_event == nil) {
                next_event = [[OEvent alloc] init];
                [next_event init_event:fp:type];
        } else
                [next_event add_event:fp:type];
}

- process : xm : (XEvent *) event
{
        if (event->type == e_type)
                return (*e_proc)(xm, event);
        else
                return [next_event process:xm:event];
}

@end

@implementation OWindow

- (void) set_root : (Window) root
{
        w = root;
        w_name = (char *)malloc(sizeof("root"));
        strcpy(w_name, "root");
        e_mask = 0;
}

- init_window : xm : (const char *) master : (const char *) my_name : (int) start_x : (int) start_y :
                (int) size_x : (int) size_y : (int) border : (unsigned long) col1 : (unsigned long) col2
{
        w_name = (char *)malloc(sizeof(my_name));
        strcpy(w_name, my_name);
        e_count = 0;
        w = XCreateSimpleWindow([xm get_display], [xm get_window:master], start_x, start_y,
                        size_x, size_y, border, col1, col2);
        next_window = nil;
        event_proc = nil;
        e_mask = 0;
        return self;
```

```
}

- (void) make_window : xm : (const char *) master : (const char *) slave : (int) start_x : (int) start_y :
                (int) size_x : (int) size_y : (int) border : (unsigned long) col1 : (unsigned long) col2
{
        if (next_window == nil) {
                next_window = [[OWindow alloc] init];
                [next_window init_window:xm:master:slave:start_x:start_y:size_x:size_y:border:col1:col2];
        } else
                [next_window make_window:xm:master:slave:start_x:start_y:size_x:size_y:border:col1:col2];
}

- (void) add_event : (Display *) dpy : (const char *) w_target : fp :(int) type : (long) mask
{
        if (strcmp(w_name, w_target) == 0) {
                if (event_proc == nil) {
                        event_proc = [[OEvent alloc] init];
                        [event_proc init_event:fp:type];
                } else
                        [event_proc add_event:fp:type];
                e_mask |= mask;
                XSelectInput(dpy, w, e_mask);
        } else
                [next_window add_event:dpy:w_target:fp:type:mask];
}

- (Window) get_window : (const char *) w_target
{
        if (strcmp(w_name, w_target) == 0)
                return w;
        else
                return [next_window get_window:w_target];
}

- run : xm : (XEvent *) event
{
        if (event->xany.window == w)
                return [event_proc process:xm:event];
        else
                return [next_window run:xm:event];
}

@end

@implementation XManager

- free
{
        XCloseDisplay(dpy);
        return [super free];
}

- (Display *) init_manager : (const char *) my_name
{
        dpy_name = (char *)malloc(sizeof(my_name));
        strcpy(dpy_name, my_name);
        dpy = XOpenDisplay(my_name);
        root = [[OWindow alloc] init];
        [root set_root:DefaultRootWindow(dpy)];
        screen = DefaultScreen(dpy);
        gc = XCreateGC(dpy, [root get_window:"root"], 0, NULL);
        return dpy;
```

```
}

- (void) add_window : (const char *) master : (const char *) slave : (int) start_x : (int) start_y :
                (int) size_x : (int) size_y : (int) border : (unsigned long) col1 : (unsigned long) col2
{
        [root make_window:self:master:slave:start_x:start_y:size_x:size_y:border:col1:col2];
}

- (void) add_event : (const char *)w_target : fp : (int) type : (long) mask
{
        [root add_event:dpy:w_target:fp:type:mask];
}

- (void) add_usual_proc : fp
{
        usual_proc = (void (*)(id, XEvent *))fp;
}

- (void) map_window : (const char *) w_target
{
        XMapWindow(dpy, [root get_window:w_target]);
}

- (void) map_windows : (const char *) w_list
{
        XMapWindow(dpy, [root get_window:w_list]);
        XMapSubwindows(dpy, [root get_window:w_list]);
}

- run_loop
{
        while (1) {
                XNextEvent(dpy, &event);
                (*usual_proc)(self, &event);
                [root run:self:&event];
        }
}

- (Display *) get_display
{
        return dpy;
}

- (Window) get_window :
                (const char *) w_target
{
        return [root get_window:w_target];
}

- (int) get_screen
{
        return screen;
}

- (GC) get_gc
{
        return gc;
}

@end
```

```
//Test program for OXlib.
//        ver0.1.
//(C)Hiroo Coshimoto
#include "OXlib.h"

#define BORDER 2
#define BOX_W 350
#define BOX_H 250
#define BUTTON_W 50
#define BUTTON_H 20

char QUIT[] = "Quit";

void usual(id, XEvent *);
void Button_Press(id, XEvent *);
void Button_Expose(id, XEvent *);
void Box(id, XEvent *);

void Button_Press(id man, XEvent *event)
{
        exit();
}

void Button_Expose(id man, XEvent *event)
{
        XDrawString([man get_display], [man get_window:"button"], [man get_gc], 5, 15, QUIT, strlen(QUIT));
}

void Box(id man, XEvent *event)
{
        XDrawLine([man get_display], [man get_window:"box"], [man get_gc], BOX_W / 2, BOX_H / 2,
                event->xbutton.x, event->xbutton.y);
}

int main()
{
        id xm = [XManager new];
        unsigned long white, black;

        [xm init_manager:""];
        white = WhitePixel([xm get_display], [xm get_screen]);
        black = BlackPixel([xm get_display], [xm get_screen]);

        [xm add_window:"root":"box":100:100:BOX_W:BOX_H:BORDER:black:white];
        [xm add_window:"box":"button":10:10:BUTTON_W:BUTTON_H:BORDER:black:white];

        [xm add_event:"box":(id)Box:ButtonPress:ButtonPressMask];
        [xm add_event:"button":(id)Button_Press:ButtonPress:ButtonPressMask];
        [xm add_event:"button":(id)Button_Expose:Expose:ExposureMask];
        [xm add_usual_proc:(id)usual];

        [xm map_windows:"box"];

        [xm run_loop];
        [xm free];
}

void usual(id xm, XEvent *e)
{
        return;
}
```

```
xbutton : xbutton.o OXlib.o
        cc -L/usr/X11R6/lib -g -o xbutton xbutton.o OXlib.o -lobjc -lX11

xbutton.o : xbutton.m
        cc -I/usr/X11R6/include -g -c xbutton.m

OXlib.o : OXlib.m
        cc -I/usr/X11R6/include -g -c OXlib.m
```

簡易型携帯電話システム概要

電子情報工学科　1回生　辰巳大佑

PHSという言葉を聞いて、皆さんはどう思われるだろうか。「スグ切れる」、「使えない」、またこれが原因で「安っぽい」、「070はカッコ悪い」、「中高生のおもちゃ」など、大抵の場合良い答えが返ってこない。それでは、なぜ安っぽくて使えないと思われるのか。その理由を検証してみようと思う。なお、ここではいわゆる090から始まる電話番号が与えられている携帯・自動車電話のことを携帯電話と呼ぶことにする。また、書かれてある内容はネット上などから得た知識が多いため、一部事実とは異なった内容が含まれているかもしれない。

1．PHSの誕生と歩み

PHSサービスが始まったのは平成6年のことである。当然ながら昭和54年に始まっていた携帯電話サービスに比べて後発であったわけでが、携帯電話がまだまだ高級な代物であったサービスイン当初は非常に大きな期待がもたれていた。そのため、滑り出しはNTTパーソナル、DDIポケット、アステル各社とも順調で、現在の携帯電話市場並みの勢いで加入者は急増した。

しかし、開発者サイドは「屋外でもつかえるコードレス電話」としての利用を想定していたのに対し、利用者サイドは「通話料が安い携帯電話」としての期待を持ってしまった。これがいわゆる「使えない」というレッテルを貼られる原因となってしまったわけである。

例えば、特に都心部で加入者の伸びに比べ回線数を十分に確保することが出来ず、発着信とも出来ない状態が頻繁に起きた。ちなみにPHSの基地局は3回線収容のものがほとんど（携帯電話は通常48回線。一部は96回線）で、ある特定の個所が著しく混雑する可能性が高いのである。ちょうど現在のJ-PHONEで見られるような現象が3社とも（とくにDDIポケット）でみられ、PHSに対する風当たりを悪くするという結果になってしまった。

さらに、高速移動中に通話が出来ないという点も問題になった。本来PHSは屋外でもつかえるコードレスホンであるので、また、電話機自体の価格を安く押さえるため高速移動中にも通話できるような設計はなされていなかった。

これらの問題は事業者サイドのPRと準備不足であり、言ってしまえば自業自得であったわけなのだが、この時点での失敗が当初の4千万ユーザー獲得目標から程遠い結果へと結びつくことになる。

さて、その後は新規契約よりも解約が多い状況がおき、契約を取る毎にインセンティブを発売店側に支払っていた各事業者はますます苦戦を強いられることとなる。アンテナ増設などの設備投資が一段落ついてからもこの現象はそれほど改善されることはなく、1998年、ついに第一号としてNTTパーソナル電話がNTTドコモに事業譲渡される。それに続きアステル東京が東京電話に事業譲渡された。事業譲渡されたところで持ち直すわけでもなく、携帯電話の右肩上がりの携帯電話加入者数に比べ、波打ったような中途半端なグラフを描きながらPHSは加入者を増やしたり減らしたりしてきた。

現在、少し状況が特殊な関西圏を除くと、高速ハンドオーバーや高速データ通信機能を搭載し、PHSの弱点を多少は克服したDDIポケットがそこそこ安定したユーザー数を獲得してい

るが、他の２社は相変わらず苦戦しているようである。また、アステル東北、中部、関西が既に、アステル九州がまもなく事業譲渡されようとしている。

２．各事業者の特徴

現在 PHS 事業を展開している PHS 事業者を紹介する。

○NTT ドコモ（旧 NTT パーソナル、NTT 系）

名前からわかるように日本の通信業界を悪い意味で独占してきた NTT が親会社の事業者である。当然ながら街中に立てられている親会社の NTT 柱に PHS 基地局を取り付けることができるため基地局の数は想像以上に多く、かなり細い路地にも取り付けられていることがある。しかし、電信柱に取り付けるわけであるからそれほど大きい設備を取り付けるわけにはいかず、１つの基地局のカバーエリアはせいぜい１００メートルであるため、とくにそのような小出力局の多い関西地区では次々とハンドオーバーがおこり、折角の高速ハンドオーバー機種でもあまり使い物にならなかったという話を聞いたこともある。NTT パーソナルをそのまま引き継いでいるのであまり PHS には力を入れるつもりはないらしく、仕方がないといえば仕方がないのであるが、独占的地位を築くことができた携帯電話業界のようにはいかず、PHS 業界では第２位に甘んじている。

NTT ドコモのに割り当てられている電話番号は 070-51**, 52**, 53** ,61**, 62**, 63**である（下４桁を除いた部分に関して）。

○DDI ポケット（KDDI、旧 DDI 系）

こちらの名前からわかるとおり、先月 DDI と KDD、IDO の合併で誕生した KDDI 系の PHS 事業者である。KDDI は「がんばれ NTT、がんばる KDDI」やら「通信業界第２位」などという今までの DDI では考えられなかったような積極的な CM を流しているとおり通信業界（NTT に大きく差をあけられ）第２位であるが、PHS 事業では第１位、それも勝ち組の感がある。しかし、そんな好調さとは裏腹に NTT と同じく、KDDI があまり PHS 事業に力を入れるつもりがないらしいのは非常に残念なことではある。NTT や電力系のように電信柱を所有していないので建物の屋上に基地局が多く設置されているが、それら基地局の出力はほとんどが０．５W の高出力タイプ、真っ先に高速ハンドオーバー対応機種の発表など、PHS 事業者の中でもどちらかというと携帯電話に近いサービスを打ち出してきたのが功を奏し、思ったほど契約数は伸びていないもののそこそこ順調に推移しているようである。最近でも積極的に基地局の増設に努めており、京都工繊大周辺にもかなりの数が、また構内にも３つのアンテナが立って、携帯電話が圏外になる場所ですら通話することが可能である。また余談であるが、現在の DDI ポケット社長、岡田健氏は京都工繊大工芸学部出身とのことである。

DDI ポケットに割り当てられている電話番号は 070-501*～509*, 54**, 55**, 56**, 60**, 64**, 65** ,66**である。

工繊大構内に設置されたDDIポケットのアンテナ（東構内事務局）

○アステルグループ（電力会社、日本テレコム系）

電力会社と旧国鉄系の日本テレコムどいう、官僚体質がプンプンしてそうな気がしてしまう会社であるが、そのためかなのか、やはりうまくいかなかった。一部の地域ではNTT回線ではなく独自の回線を利用して他の２事業者とは違う形で事業を開始したのだが、高出力アンテナを導入しなかったこと、エリア化の方法が中途半端であったなどの理由で次々と利用者離れを起こし、北海道、東北、東京、中部、さらには最も好調と思われた関西とほとんどの会社が事業譲渡される結果になってしまった。また、近いうちに九州も事業譲渡されるとのことである。関西地区では電信柱（関電柱）の上に取り付けられたロケット型のアンテナが特徴的である。

アステルグループに割り当てられている電話番号は070-500*, 57**, 58**, 59**, 67**, 68**, 69**である。

３．携帯電話との違いなど

携帯電話とPHSの特徴を表にまとめたものを下に示す。

	携帯電話	簡易型携帯電話(ＰＨＳ)
電話機の出力	０．１～０．８ワット	０．０１～０．０８ワット
基地局の出力	約４ワット	０．０２～０．５ワット
１つの基地局のカバー範囲	半径５キロ程度	半径５００メートル程度（０．５Ｗの場合）
基地局の設置費用	数千万～1億円程度	100万円程度
利用者数	約５６００万人(2000/9)	約５７０万人(2000/9)

この表を見ていただいて分るように、一言で言ってしまえば PHS はすべてが簡易に作られているのである。しかも、ほとんどの値が 10 分の 1 になっているのが面白い。

携帯電話と PHS との最大の違いは、電話機や基地局の出力の違いである。出力が端末、基地局ともに携帯電話の 10 分の 1 しかない PHS では、当然ながら基地局がカバーする半径も小さくなってしまう。半径が 10 分の 1 であれば面積はさらに差が広がってしまうわけであり、いくら基地局の設置費用が安いといっても人口の少ない地域をくまなくサービスエリア化することは非常に困難である。ユーザ数が多く、数多くの基地局が設置されている都心部ならともかく、郊外ではすこし街から離れただけで圏外になってしまうのはこのためである。とは言うものの、現在ではかなりの場所がサービスエリア化されているのだが、都心部ですらすこし中心から離れたら使えないという現象が起きていたサービスイン当初のイメージを未だに引きずっているようである。

その他の要因としては、端末が安っぽいというのもあるのかもしれない。最近の携帯電話新機種の液晶画面はカラーのものが非常に多くなってきた。今や 4 事業者すべてからカラー端末が出ているのであるが、PHS 事業者からは 10 月末現在、まだ 1 社も発売されていない（発表はされている)。また、現在の携帯電話機ではあたりまえとなった大型液晶を搭載した機種すら皆無に近い状態である。

そして、決定的な要因としては、メール機能の貧弱さが挙げられる。携帯電話事業者では１２８文字１円で受信料無料、など非常に魅力的なメールサービスを提供しているのに対し、PHS 事業者では未だに送受信とも７円や１０円、など、非常に使いづらいサービスしか提供していない。３千文字利用可能など長文のメールを受信する人にとってはありがたいのだが、最近では「おはよう。今日、暇？」とか「おやすみ。今日はお疲れ様」程度の文字数のメールしか送らない人が大半であるので、１通に１０円も取られる PHS が嫌われてしまうのは無理もないのかもしれない。

何か悪いことばっかり書いてしまった。もちろん悪いことばかりであるのならば私も他の人も利用しないだろう。当然ながら素晴らしい点も多い。

たとえば、データ通信速度の速さ、通話品質の良さが挙げられる。これは携帯電話より高い周波数(1.9GHz、携帯電話は 800MHz または 1.5GHz)を利用していることと、携帯電話とは違い帯域に比較的余裕があることなどから実現できているのである。通話品質に関してはハーフレートと呼ばれる聞き取るのがやっとであるぐらいの音質が一部で使われているの携帯電話に対し、PHS は固定電話に近い音質を実現している。

さらに、基本料、通話料が安いという点が上げられる。最近では携帯電話の基本料や通話料も安くなってきているが、昼間の市内通話に関しては PHS に遠く及ばない。これは PHS の仕組みが基本的に ISDN 回線を利用していることによるもので、加入電話並とは行かないものの携帯電話よりは安い通話料を実現しているのである。(ただし、長距離通話に関してはかなり高額となり、携帯電話のほうが安くなってしまうが)。

もう一つ、電池の持ちが良いという事もあるかもしれない。10 分の 1 の出力しかないので、さすがに 10 倍とは行かないものの 5 倍程度は持つようである。ちなみに私は 2 週間程度充電しなくても使えたことがある。携帯電話ではそうは行かないだろう。

そして、出力が小さいのを逆手にとって、利用者が浴びる電磁波の量が少ないということもある。実際に医療機関の施設の中でも医療用 PHS（出力は同じ）が利用されているぐらいで、携帯電話とは違い電子機器の誤作動はまずないであろう。

4．今後 PHS はどうあるべきか。

以上に挙げたとおり、PHS 業界は現在非常に苦しい立場に立たされている。では、どうすれば携帯電話並とは言わずともユーザ数を伸ばすことができるのだろうか。

第一に、もっと興味を引くような CM が必要である。今度の DDI ポケットの CM キャラクターは梅宮アンナらしい。私が好きかどうかはともかくとして、いったいどの層を狙っているのだろうか？前回のトータス松本+江口洋介もいまいち解らなかったが、どの層にもうけるキャラクターを使うわけでもなく、かといって特定の層を狙ったわけでもないような CM を流すのは、金の無駄づかいである。もともと PHS 各事業者は携帯電話事業者に比べるとほとんど CM を流していない（ドコモ、アステルは論外の少なさ）に等しいが、流すにしても下手過ぎる物しか流れていない（DDI→KDDI もそうだが）。いっそ PHS がまだ元気だったころのように、おバカなタレントをを採用しておバカな CM を流すべきである。

次に、高級感のある機種の発売も挙げられる。お分かりの方は良くお分かりだと思うが、未だに PHS は安っぽい機種しか出てこない。最近の携帯電話と比べたら一目瞭然で、まず外側のカバーの素材からして安っぽい（というか、柔らかそうな素材はなんとなく安っぽく見える）。携帯電話を落として故障した、という話は良く聞くのだが、PHS を落として蓋が開き基盤が見えたが、パチンと戻したら直った、という話を聞いたことがあるので安っぽい素材にもそれなりの利点があるのかもしれないが、やはり携帯電話のように魅力のある機種が欲しいところである。

最も重要なのが、通話料金の引き下げである。PHS では原則として基地局より先は NTT 回線であるため通話ごとに 10 円の接続料が加算される（例外はある）。これがあるために１５秒程度の通話では携帯電話より割高になってしまう可能性がある。この接続料に関しては NTT の言いなりで、NTT が赤字ならば不足分を請求し、黒字であればそのまま、という噂（あくまでも噂だが）もあり、事業者側だけではなんともならないのかもしれないのだが、なんとかしていただきたいものである。

そして、更なる通話可能エリアの拡大、つまり基地局の増設である。実はこれも、NTT からの言いなりでなかなか解説できないという事情もあった（ある？）らしい。（だから私は NTT が嫌いなのだが、そういう話をすると本題から外れるのでやめる。）NTT 独占の弊害が PHS の発展を邪魔してきたのかどうかは知らないが、基地局設置が完了しても開通するのが遅れていた時期があったのは確かで、これからは工繊大に基地局が設置されたように積極的な展開をお願いしたいものである。

私は PHS ユーザであり、この業界の方が近くにいることもあり、また NTT は好きでないため KDDI 系ががんばっている PHS が好きである。しかしこのままではポケットベルのような運命をたどってしまう可能性がある。折角のすばらしい技術を無駄にしないためにも、またこれ以上携帯電話から発せられる強力な電磁波を街中に撒き散らさないためにも PHS にはがんばってもらいたいと思う。

Windows Tips

電子情報工学科　1回生　春井宏介

Windows レジストリを中心に使用するカスタマイズを紹介していきます。現在のご自分の環境にもうひと味付けたい方にも役立てば幸いです。

◇一般環境カスタマイズ◇

スタートボタンの右クリックメニューに項目を追加

[HKEY_CLASSES_ROOT¥Directory¥shell]内に新しいキーを作成します。キー名は自由に付けて構いません。作成したキーの（標準）の値にメニューとして表示させる名前を書きます。さらに、そのキーの下に command という名前の新規キーを作成し、command キーの（標準）の値に開きたいアプリケーションのフルパスを入力します。

スタートメニューの表示を早くする

[HKEY_CURRENT_USER¥Control Panel¥Desktop]内に新しい文字列 MenuShowDelay を作り、値のデータを変更（1［最も速い］）します。ただし、早くしすぎるとマウスを移動する際にすべてが表示されていくので逆にパフォーマンスが落ちることもあります。

ショートカットを作成した時に「～へのショートカット」をなくす

[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explore]の link のバイナリ値を 00 00 00 00 にします。

ショートカットの矢印の削除

[HKEY_CLASSES_ROOT¥SOFTWARE¥Classes¥lnkfile¥IsShortcut]
と
[HKEY_CLASSES_ROOT¥SOFTWARE¥Classes¥piffile¥IsShortcut]
の二つのキーを削除するとショートカットアイコンに付く矢印が表示されなくなります。

デスクトップ上のシステムアイコン関連の操作

まず、それぞれのアイコンのアドレスを書いておきます。
Microsoft Outlook（旧「受信トレイ」）：
HKEY_CLASSES_ROOT¥CLSID¥{00020D75-0000-0000-C000-000000000046}
デスクトップ：
HKEY_CLASSES_ROOT¥CLSID¥{00021400-0000-0000-C000-000000000046}
ネットワークコンピュータ：
HKEY_CLASSES_ROOT¥CLSID¥{208D2C60-3AEA-1069-A2D7-08002B30309D}
マイコンピュータ：

HKEY_CLASSES_ROOT¥CLSID¥{20D04FE0-3AEA-1069-A2D8-08002B30309D}
コントロールパネル：
HKEY_CLASSES_ROOT¥CLSID¥{21EC2020-3AEA-1069-A2DD-08002B30309D}
ごみ箱：
HKEY_CLASSES_ROOT¥CLSID¥{645FF040-5081-101B-9F08-00AA002F954E}
Internet Explorer：
HKEY_CLASSES_ROOT¥CLSID¥{871C5380-42A0-1069-A2EA-08002B30309D}
上記のアドレスを利用して次のようなカスタマイズが出来ます。
マイコンピュータのアイコンを変更するには上記の場所の DefaultIcon のデータを変えます。
スタートメニューにコントロールパネルを追加するには、[C:¥WINDOWS¥ｽﾀｰﾄ ﾒﾆｭｰ]に「コントロールパネル.{21EC2020-3AEA-1069-A2DD-08002B30309D}」という名前の新規フォルダを作成します。
ネットワークコンピュータの名前を変更するには、上記の場所の（標準）文字列を、たとえば「マイネットワーク」と変更します。
マイコンピュータのアイコンにマウスを乗せた時に表示されるヒントを編集するには、上記の場所の[InfoTip]文字列を書き換えます。

スタートメニューから「検索」「お気に入り」などを隠す

[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Policies¥Explorer]内の「No…」の DWORD 値を 1 にします。（ご自分で分かる「No…」の変更を行って下さい。）

ウィンドウのアニメーションをオフにする。

[HKEY_CURRENT_USER¥Control Panel¥desktop(¥windowmetrics)]内の MinAnimate （なければ新規に作成する）をダブルクリックし、値のデータを 0（オフ）にします。

プログラムメニューをアルファベット順に並べる

Windows95+IE5 や Windows98 以降の環境ではプログラムメニュー内での右クリックメニューにより「名前順に並び替え」が出来るようになっていますが、新規に項目を追加するとその項目は一覧の最後に登録されてしまいます。Windows95 の時のように何もしなくても名前順になってほしいという希望を解決しましょう。
[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥MenuOrder¥Start Menu¥&Programs¥Menu]の Order キーを削除して Windows を再起動すれば順序情報を持たないさっぱりしたプログラムメニューになります。これは上記のキー内のアプリケーションごとのメニューでも同じであり、また、
[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥MenuOrder¥Favorites]でも通用します。ただし、どれも一度でもメニューの並べ替えをすると Order キーが作成されてしまうので注意しましょう。

インターネット使用時のデータの転送速度を上げる

さまざまな制限を変更することでインターネットを快適にすることが出来ます。

[HKEY_LOCAL_MACHINE¥System¥CurrentControlSet¥Services¥Class¥Net¥000x] キーを開きます（x は数値で、[DriverDesc]エントリに“TCP/IP”が登録されているキーを指定します)。[編集]→[新規作成]→[文字列]を選択して[IPMTU]エントリを作成します。[IPMTU]エントリを選択して[値のデータ]に‘1500’と入力する。これにより 1 個のパケットで送受信できるデータサイズが大きくなります。さらに、

[HKEY_LOCAL_MACHINE¥System¥CurrentControlSet¥Services¥Class¥NetTrans¥000x] キーを開きます（x は数値で、[DriverDesc]エントリに“TCP/IP”が登録されているキーを指定します)。[編集]→[新規作成]→[文字列]を選択して[MaxMTU]エントリを作成します。[MaxMTU]エントリを選択して[値のデータ]に、

アナログ回線接続なら‘576’

ISDN 回線接続なら‘1152’

と入力します。これにより伝達可能なパケットサイズが大きくなります。さらに、

[HKEY_LOCAL_MACHINE¥System¥CurrentControlSet¥Services¥VxD¥MSTCP] キーを開き、[編集]□[新規作成]□[文字列]を選択して[DefaultTTL]エントリを作成します。[DefaultTTL]エントリを選択して[値のデータ]に‘64’と入力します。これによりネットワーク的に遠過ぎるサイトでも到達し易くなります。

◇小テクニック◇

デスクトップアイコンの削除

[HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥explorer¥Desktop¥NameSpace] の中で消したいアイコンのキーを削除します。

壁紙を中央以外の好きなところに貼り付ける

[HKEY_CURRENT_USER¥Control Panel¥desktop]内の

「WallPaperOriginX」と「WallPaperOriginY」の値に（ない場合は新規に作成する）それぞれ、壁紙を表示したい座標を入力します。入力する値は画面の左上からのピクセル数を指定して下さい。Windows を再起動すれば完了です。

アイコンの表示色を変更する

アイコン表示に使える色数を変更します。

[HKEY_CURRENT_USER¥Control Panel¥desktop¥windowmetrics]内の Shell Icon BPP の値を変更します。ハイカラーなら 16、フルカラーなら 25 または 32 にします。設定後、Windows を再起動させます。

アイコンの表示異常を修復する（再起動しても直らない場合）

Windows はアイコン表示高速化のためアイコンデータのキャッシュを作っています。これを初期化します。C:¥Windows¥ShellIconCache を削除して再起動すれば完了です。いったん Safe Mode で Windows を起動しても同等の効果が得られます。

不必要なスタートアップを取り消す

[スタートアップ]以外のスタートアップアプリケーションを中止する方法です。msconfig.exe を起動して「スタートアップ」タブで不要な物のチャックを外しましょう。ただし、分からない項目についてはチャックを外さないこと。Windows が正常に動作しなくなるだけでなく、起動すらしなくなる可能性があります。レジストリに登録されているスタートアップは [HKEY_LOCAL_MACHINE¥Software¥Microsoft¥Windows¥CurrentVersion¥run] にその情報があります。完全な削除を希望する場合はこちらを操作してください。

「ファイル名を指定して実行」の履歴の削除

[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥RunMRU]の（標準）以外のキーを削除します。

不要なプログラムメニュー情報の削除

[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥MenuOrder¥Start Menu¥&Programs]には削除したはずのプログラムメニューが残っていることがあります。Windows の動作安定化に多少とも貢献するため不要な物は削除しておきましょう。

デフラグの動作条件変更

[HKEY_LOCAL_MACHINE¥Software¥Microsoft¥Windows¥CurrentVersion¥Applets¥Defrag¥AppStartParams]内で以下の設定が出来ます。
アプリケーションログの使用…UseProfile の DWORD 値（ない場合は作成）を変更する（0 なら使用しない／1 なら使用する）。
最適化しないファイル…ExcludeFiles の値（文字列）を変更します。

再起動せずに SCSI デバイスを利用したい

外付け SCSI 機器の電源を入れずに Windows を起動したが、その SCSI を使いたくなったとき再起動の手間に嫌になることはないでしょうか。この手間は次の操作で大幅に省けます。デバイスマネージャー（「コントロールパネル」の「システム」、またはマイコンピュータのプロパティ）で「更新」をクリックします。この作業によって SCSI デバイスが認識され、再起動の必要なく起動していなかった SCSI 機器を使えることになります。

CD-ROM のオートランを修復する

デバイスマネージャーの CD-ROM のプロパティで自動起動にチェックは入っているのにも関わ

らずCD-ROMがオートランしなくなるトラブルがあります。
[HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Policies¥Explorer]のNoDriveTypeAutoRunの値を 95 00 00 00にするとこの問題は解決します。(ff 00 00 00になることがあるようです。)

Microsoft Outlookへのショートカット

[outlook]がOutlookへのショートカットアドレスですが、[outlook:¥¥個人用フォルダ¥受信トレイ]とすればOutlook起動時に受信トレイが開きます。「ファイル名を指定して実行」でも使えますし、ショートカットアイコン作りに利用するのもいいでしょう。

◇メモリ(キャッシュメモリ)設定◇

主な使用目的を「ネットワークサーバー」にする

[システムのプロパティ]→[パフォーマンス]→[ファイルシステム]で参照されるコンピュータの使用目的の欄は、一般的には「デスクトップコンピュータ」になっています。この場合、キャッシュメモリ 10KB・32 のディレクトリパス(PathCache)・677 のファイル名を記録できる(NameCache)設定となっています。これを「ネットワークサーバー」に設定するとキャッシュメモリ40KB・PathCacheが64エントリ・NameCacheが2729エントリになります。この変更をすることにより、キャッシュ効果の大きい環境でPCが利用できます。

VCACHEの容量を固定する

SYSTEM.INIの[vcache]に
MinFileCache=最小サイズ(1)
MaxFileCache=最大サイズ(2)
を追加します。ただし、この変更はWindows98以上の環境では効果が見られないことが多いようです。
<参考:メモリ64MB以上を想定>
・平均5以上のアプリケーションを起動して使用するユーザ向け(パフォーマンスアップ小)
(1)=2048 (2)=20480
・平均4以内のアプリケーションを起動して使用するユーザ向け(パフォーマンスアップ大)
(1)=10240 (2)=10240 (VCACHE固定化)

【備考】

本文中に紹介したのは、Windows 9x系OSが前提のTipsです。システムを破壊するようなものは載せていませんが、極力、システムやレジストリのバックアップをとってから、あくまで貴方の責任で操作して下さい。また、復旧が困難に思えるものについては必ずバックアップをお取り下さい。

Web掲示板（と、チャット）を考える

1回生　松村 宗洋

コンテンツ：

1、CGIとはなんだろうか
2、CGIを使った掲示板、（チャット）の仕組み
3、掲示板CGIプログラムの詳細な処理過程
4、掲示板（と、チャット）の安全を考える

※プログラム的な解説は一切ありません。どうかお気楽にお読みください

1．CGIとはなんだろうか

近年急速に発展してきた、個人Webサイトにおいて欠かせなくなってきた、掲示板システム（BBS＝Bulletin Board System とも言う）や、チャットルーム(Web上で同時に入室しているメンバーが発言文を投稿しあって、会話をする場) がネットサーフィンをしていると、至るところでみかけるようになりました。この掲示板やチャットといった動的にWeb上のHTMLを変えることができる技術に『CGI』というものがつかわれています。これは『Common Gateway Interface』の頭文字をとったもので、Web上でデーターベースなどの文書を検索したり、保存、ブラウザが見えるHTMLとして出力したりするプログラムの総称を言います。

CGIについての突っ込んだ内容を書くと分厚い本が出来上がるので、ここではCGIの詳しい説明は避けて、主に前述した「掲示板」、「チャット」がどのような仕組みで動いているのかを、わかりやすく説明したいとおもいます。以下の記事を読むにあたって、プログラムの知識は必要ないと思います（たぶん・・・）

この記事を読んでいる方はすくなくとも「HTML」という単語をご存知かと思います。これは、ホームページなどのWebサイトで現在主に使われている文書の記述言語です。このHTMLをつかえば、素っ気ないただのテキストを、豪華に装飾してくれます。そして見栄えのするホームページとして公に公開することができるのです。Web ページを使えば、いままで一部の人間にしか扱えなかった「情報」というものが、個人レベルで発信することが可能となりました。しかし、このHTMLで書かれた「文書ファイル」はただ単に「文書」でしかなく、作成者の一方的な情報発信源でしかありません。その文書ファイルのあるWebサイトの閲覧者、訪問者は、ただそれを見て通りすぎるだけなのでした。そこで、作成者と閲覧者とのインタラクティブ性を図ったものが登場しました。それが「掲示板」です。このシステムの登場によって、そのサイトを閲覧した誰もが「書き込み」を行うことができるようになりました。そして、このシステムを特化したのものに「チャット」というものが出現しました。チャットルームに入っている誰もが、「書き込み（発言）」をして、会話が成立するという驚くべきシステムです。会話といった意思疎通が成り立つことによって、またそこに一つの社会が誕生します。これが早い話、「ネット社会」というもので、その内容に進むとまた趣旨から外れることとなってしまうので、このあたりでとめておきます。言いたいことは、CGIを使ったWeb上でのプログラムの実行（Webアプリケー

ション）によって、掲示板やチャットといったものが成り立っているのだということを、わかっていただけたらいいのです。

2、掲示板、チャットの仕組み

さて、ここで本格的に掲示板の仕組みに移っていきたいと思います。まず、掲示板に書き込みをする、という動作を細かく見てみることにしましょう。

早速ですが、掲示板にこれから書き込んでみましょう。とある掲示板（実は私がこの原稿執筆時に作成中のものだが）にブラウザを使って、アクセスしてみました(図１)。

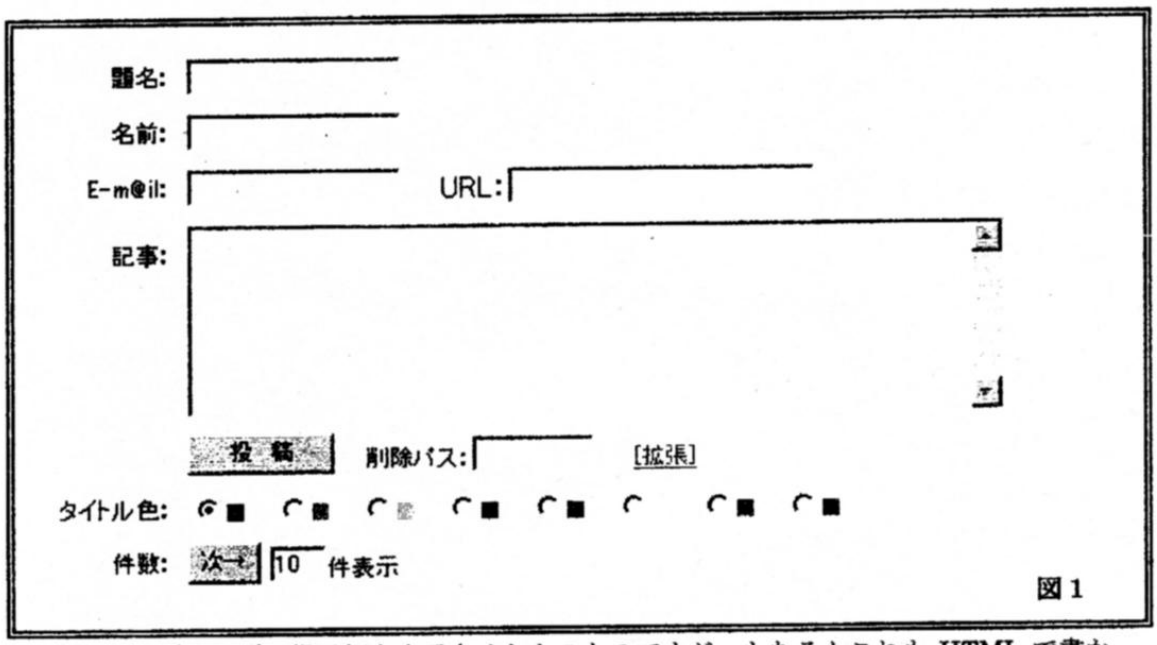


図１

図１はブラウザで掲示板を表示させたものなのですが、もちろんこれも HTML で書かれています。そのソースコードは以下のようになっています。[一部略]（図２）

```
<form action="./record.php3" method=POST name=f1>
<table border=0 cellpadding=4 cellspacing=0 align=center width=100%>
<tr><td width=20% align=right><font size="-1">題名:</font></td>
<td> <input type=text name=Title1 value="" size="20"> </td>
</tr><tr>
<td width=30% align=right><font size="-1">名前:</font></td>
<td> <input type=text name=Name1 value="" size="20"> </td>
</tr><tr>
<td width=30% align=right><font size="-1">E-m@il:</font></td>
<td> <input type=text name=Email1 value="" size=20>
URL: <input type=text name=Url1 value="" size=30>
</td></tr><tr>
<td width=30% align=right><font size="-1">記事:</font><br><br><br><br><br></td><td>
  <textarea rows=7 cols=60 name=Kiji1></textarea>
</td></tr><tr><td width=30% align=right> </td>
<td> <input type=submit name=Submit1 value=" 投 稿 ">
<font size="-1"></td></tr><tr>
          .
        (略)
          .
</td>
</tr>
</table></form>
```

図１において、テキストを入力する部分を「フォーム」といいます。説明を簡略化するため、図１のには「題名」「名前」「E-mail」「URL」「記事」の入力欄があるということにしましょう。そして掲示板に書き込むための入力をしました（図３）

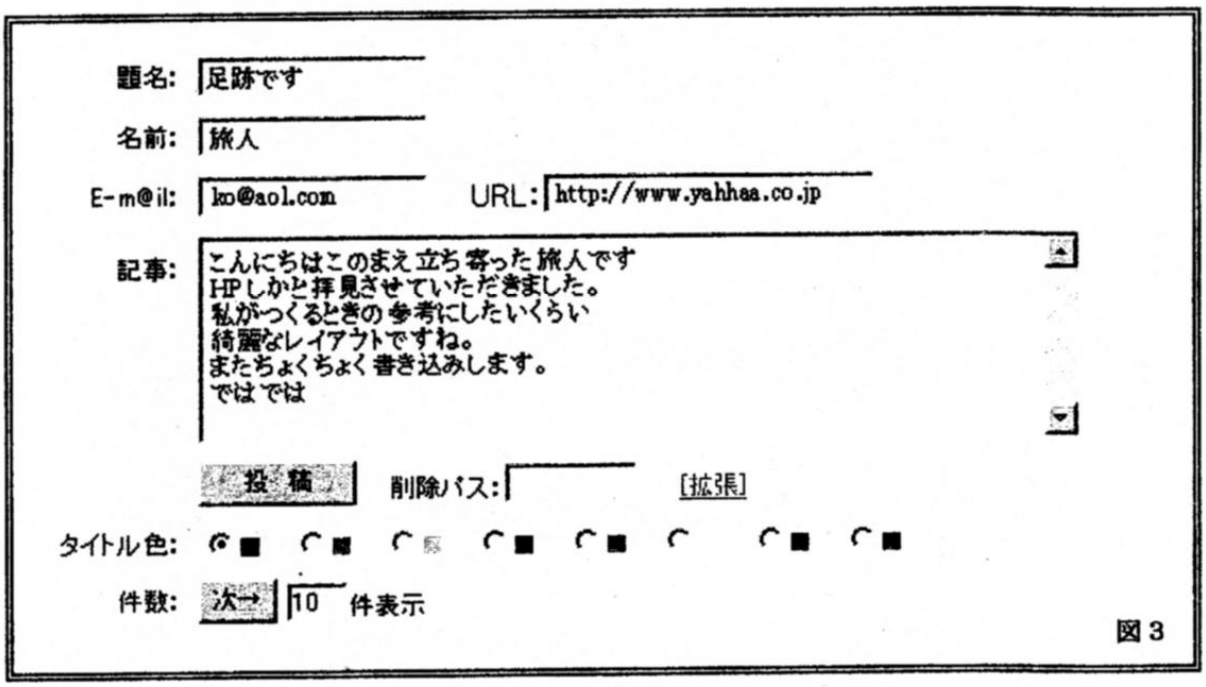


図３

図３のように書き込みをして、あとは［ 投 稿 ］ボタンを押せば書き込みはできます。さて、ここで見てもらいたいのは、図２です。図１においての 「題名」「名前」「E-mail」「URL」「記事」にあたる入力フォームのソース部分を図２で ***斜体*** で書いてあります。それぞれのフォームに名前がつけられていることがわかりますね。それぞれのフォーム要素には以下のように名前がつけられています。

name=*Title1* ←「題名」のフォーム名
name=*Name1* ←「名前」のフォーム名
name=*Email1* ←「E-mail」のフォーム名
name=*Url1* ←「URL」のフォーム名
name=*Kiji1* ←「記事」のフォーム名

そして、図３のように入力した内容は、それぞれのフォーム名の「value（値）」という要素に格納されます。そして、一つのフォームにつき以下のような形で送信されるのです。

例：「題名」のフォームの送信内容

Title1=足跡です

それぞれのフォームの内容は上のような形となって、すべて「&(アンパサンド)」の記号で結合され、最終的に送信される内容は以下のようになります。

Title1=足跡です&Name1=旅人&Email1=ko@aol.com&Url1=http://www.yahhaa.co.jp&Kiji1=こんにちは・・・・

このようにして、どんなにたくさん入力フォームがあっても、投稿したメッセージは上

のような１行になって送信されます。つぎに、このように投稿された一行をCGIプログラムが受け取り、プログラムが動き出します。以下に書き込みをしてから書き込まれるまでのの流れを図示しました。（図４）（即席手書きで申し訳ない）

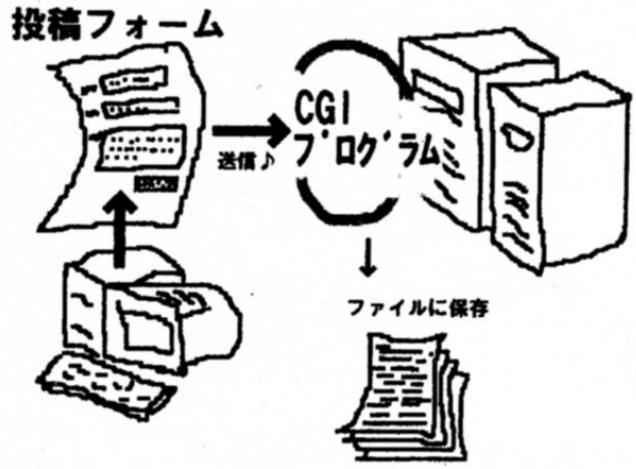


図４

そのまんまですね(笑)。このようにCGIプログラムが送信された、先ほどの一行の文字列を解読します。そして、その内容を文書ファイルとして保存します。これで掲示板への「書き込み」が完了したわけです！（※簡単に述べていますが、実際はファイルに保存するまで、多くの処理をしています）

しかし、これだけでは書き込みはできるものの、書いた『記事』を読むことができません。読むのにもCGIプログラムにアクセスしなければなりません。（掲示板によっては、書いた内容を整形してそのままHTMLファイルとして保存し、ただちに読めるようにしているタイプのものもあります）　同じように掲示板の記事を読むときのプロセスを図示しました。（図５）

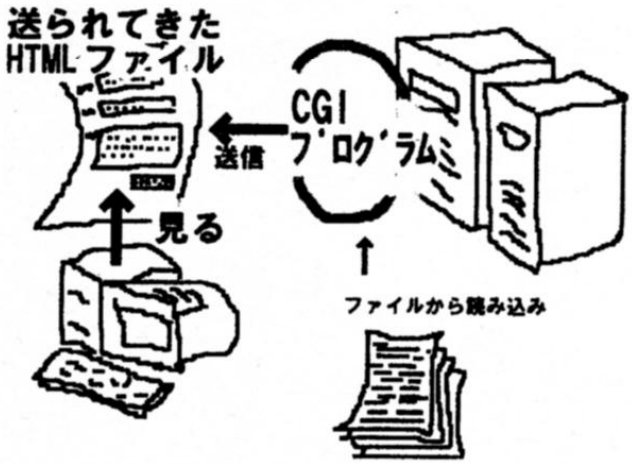


図５

ブラウザがCGIプログラムにアクセスすると、CGIプログラムは保存してある記事ファイルを読み込み、その内容を整形して、HTML文書としてアクセスしてきたブラウザに返します。これで、ブラウザはただのHTML文書ファイルをみるのと同じように掲示板の記事を読むことができるのです。掲示板やチャットはCGIプログラムのこのような「書き込み」と「読み出し」の繰り返しによってのみ、成り立っています。チャットルームのCGI

プログラムは、掲示板においての書き込み、読み出しの頻度を、会話が成立するくらいの早さでこなしている、というだけのことです。

以上で、掲示板とチャットの稼動している仕組みというのを超〜〜×３くらい簡単に説明しました。しかし、実際のところもっと詳細にどのような処理をしているのか？というのを知りたい方々のため、すこし紙面を割きたいと思います。

３、掲示板 CGI の詳細な処理過程

この章では、実際の一般的な掲示板がどのような処理をして稼動しているかを詳細に説明使用科と思います。先に断わっておきますが、現在日本で普及している掲示板スクリプトはセキュリティー的な理由から、かなり進歩を遂げたものとなっています。しかし、その分多くのプログラム的なルーチンを通っているため、以下で述べるような詳細（とは言えないが）な説明では現行のセキュリティーの需要には耐えられません。

それでは、以下のような掲示板の CGI プログラム（図６）を例にとって説明します。

Bbs.cgi	←CGI プログラム本体
Bbs.dat	←記事の保存ファイル

★書き込み

書き込みをするとき、１章で説明したように、必要な入力フォームに値を入力して、[投稿]ボタンを押します。これの動作を Submit　といって日本語に訳すると「提出」を意味します。その意味の通り、フォームの内容をすべて一行にまとめて「提出」するわけです。

以下から、CGI プログラムのする処理を説明することとなります。Submit された一行の文字列は、CGI プログラムのなかで、それぞれのフォームの要素に分割されます。簡単に説明するため、以下のような提出されたデーターを考えます。

Name1=ABC&Name2=XYZ&Name3=STU

まず、CGI プログラムは「&」記号でくっつけられたこの文字列を同じ「&」記号でそれぞれのフォーム要素に分割して、もとの状態にします。つまり、このように

Name1=ABC　　Name2=XYZ　　Name3=STU

バラバラの状態に戻します。いまそれぞれの要素は

フォームの名前＝入力された値

といった状態であるので、CGI プログラムは、今度はこれらの要素それぞれについて、値を格納する箱のようなものを仮想的につくります。そしてその箱のなかに入力された値を格納していきます。（図６）

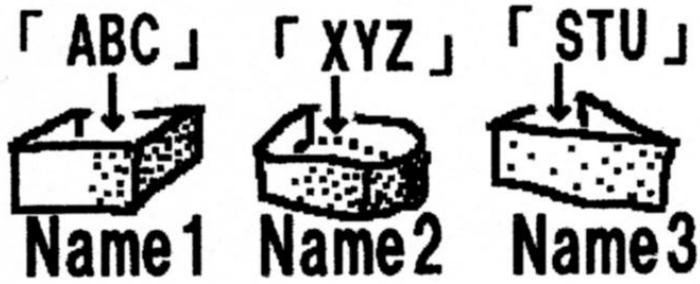


図６

これは、プログラム的に言うと、「変数に値を代入する」ということです。入力されたデーターはこの「変数」という箱のようなものの中に収められ、CGI プログラムの中で扱われます。

以下、このように変数に収められたデーターを CGI スクリプトは処理していきます。

1、文字コードの統一
2、改行コードの除去、もしくは付加
3、HTML のタグがデーターに含まれていたときの、タグの除去、置換の処理
4、入力されたデーターの HTML 装飾

1～4のように箱に入れられたデーターは車の組みたて過程のように、つぎつぎと CGI プログラムによって改竄されていきます。そして、記事の文章ファイルとして保存するのに最も適切な形となります。残るは記事の保存です。保存する記事のフォーマット（形式）は掲示板プログラムの作者が勝手に決めるものなので、以下のような方法はほんの一例にすぎません。Bbs.dat という名前で保存する記事ファイルは 1 つの記事が改行で区切られているとしましょう。このような記事フォーマットにしました。

[Name1] , [Name2] , [Name3]

コンマで区切ることにしました。Name1 に名前、Name2 に題名、Name3 には記事内容がそれぞれ当てはまります。これを 1 記事単位として、[Name3]の後ろで改行して、次の記事をその下に同じフォーマットで書き足していけるようにします。

フォーマット通りに箱を並べたら、あとは Bbs.dat　を開いて、書き込みをすれば記事の保存は完了しました。

★記事の表示

記事の表示は、書き込みでしたことと途中まで逆にしただけのようなものです。まずは、先ほど保存した記事ファイルを開きます。開いたら、この記事のそれぞれの要素は「，」で区切られているわけですから、これは、フォームから入力された文字列を「&」でわけたときと同じになります。これを今度は、HTML ファイルとして整形して、プログラムのループ文をつかって、1 記事 1 記事を書き出し、掲示板の HTML を完成させます。それを、ブラウザに送信して記事が私たちに見やすい形となるのです。

全然詳しい説明になってませんね。どうか御許し願います。

5　掲示板チャットのセキュリティーを考える

さてさて、このような自由な書き込みができるチャットや掲示板が、一つのネット社会を創るということは前述した通りです。社会がある以上、犯罪やそれに結びつくような悪戯が沢山もたらされるでしょう。とりわけ、日本のＢＢＳでは、悪戯として、「誹謗中傷」「掲示板荒らし」というものが掲示板システムのできた当時から流行ったものです。現行

の日本の掲示板は、そう簡単に荒らすことはできません。荒らしたとしても、すぐにばれてしまうという社会的セキュリティーが大きく守っているからです。それでは、ここで掲示板などを攻撃するときに用いられる手法をいくつか述べます。

1、 タグ攻撃（視覚的、物理的被害）
2、 ソース書き込み・大量書き込み攻撃（精神的な被害）
3、 ＳＳＩコード書き込み攻撃（物理的被害）
4、 誹謗中傷・罵倒攻撃（精神的被害）
5、 Ｄｕｋｅ（精神的・物理的被害）

ざっと大別するとこのくらいあります。それぞれの攻撃に対する対策を考えたいと思います。

1、タグ攻撃

掲示板といえども、閲覧者にとってはＨＴＭＬファイルをブラウジングしているにすぎません。すなわち、ブラウザが異常な状態になるようなＨＴＭＬタグを記事として書き込みすれば、ほかの訪問者は掲示板のページを表示した瞬間ハードディスクがクラッシュというようなことも起こりかねません。具体的にはどのようなタグ攻撃の種類があるのでしょうか。

・
という、書き込みをすると、という閉じタグがないので、記事の最後まで文字のサイズを 9999999999 ピクセルで表示させようとします。ばか正直なブラウザは本当にこのサイズで表示させようとするので、処理速度が追いつかなくなって、OS がフリーズしてしまいます。

・URL 書き込み欄に
"><a href="javascript:for(){alert("アホ馬鹿");}">
現行の掲示板では、記事の HTML のタグに相当する記号「<」「>」を「<」「>」といった、ブラウザが表示に問題のない記号に置き換えしまうという処理を普通はするので、1 つめの例であげた攻撃方法は実際は通用しません。しかし、抜け穴があるのです。URL 入力欄などの、普通記事を投稿しない欄に上のような書き込みをすると、タグ回避の処理をしていないことが多く、「alert("アホ馬鹿")」永遠に実行されてしまいます。見た人はもう強制終了をするしか、窓を閉じることはできません。

・<img src="mailto:ahoboka@hehhe.net"> のタグをいっぱい書き込む
＜IMG＞タグ（画像の表示のタグ）に、本来は画像のアドレスを書くところに「mailto:…」のようなメーラーの呼び出しの記述をしておきます。するとこのページを表示した訪問者はいきなりメーラーが腐るほど起動して、パソコンがハングアップしてしまいます。

・<table><tr><td><tr><td><tr><td>…</table>

のようにテーブルのタグを腐るほど書きます。

ブラウザはテーブルタグでかかれたHTMLを内部バッファですべて保存してから一気にかき出すので、これだけ沢山<Table>タグがあると、テーブルを構築するのをブラウザが失敗してしまい落ちてしまいます。この手のタグかきこみ悪戯は例をあげるときりがありません。とにかく、ほかの訪問者が見て掲示板のレイアウトをがらりと崩してしまうようなタグが書き込めれば荒らしは完了です。

★ 対策

掲示板のあらしかたばかり例をあげてもどうしようもありません。現行の掲示板はこのようなタグあらしには、以下のような対策をしています。

・とにかくタグはすべて回避するため置き換えてしまう

→とにかくこの方法が一番確実でしょう。タグが使えない掲示板でタグ攻撃をすることは不可能なのですから。しかし、タグのまったく使えない掲示板も素っ気無く、客集めには不向きかもしれません。

・記事をテーブルタグで囲んだ形でＨＴＭＬ表示させるようにする。

→これは、もし万が一<font>のような閉じタグがあるタグの閉じタグを書かなかった（書き忘れた）場合、それ以降の記事の色が全部同じになってしまった、などというトラブルを防いでくれます。<table>タグで囲まれた外には、たとえタグの閉じ忘れがあっても、</table>の部分で食い止めて表示してくれます。

・特定のタグしか使用を許可しない

→これは掲示板のCGIプログラム上での処理の話です。CGIプログラムにあらかじめ、使ってよいタグだけを記述しておき、記事が投稿されたとき、特定のタグ以外のタグはすべて、タグ回避置き換えをしてしまい表示させるとよいでしょう。

2、ソース書き込み、大量書き込み攻撃

掲示板を荒らす、といったら定番の攻撃がこの攻撃です。なんといってもその理由は「へたれな厨房もコピーしてボタンを押すだけで可能」「技術的なことはまったく必要がない」（なんだかどっちも同じ意味に…^^;;）だからです。どのような攻撃かというと、そのページのソースコードをそのままコピーして、掲示板に記事として投稿したり、大量の、見るのもウザったくなるような記事を何投稿もしたり、同じ内容の記事を何件も投稿したり、といった非常に低レベルなものばかりです。この手の攻撃の共通性として、「記事を読んでもらう意思がない」ということです。この手の書き込みにはどのような対処が必要なのでしょうか。

・ダブルポスト対策をする

→これは正規な（？）投稿者が回線の速度が遅いために、誤って2回以上投稿ボタンを押してしまうといった場合にも対処できます。これの CGI プログラムのルーチンは、新しく投稿されてきた記事が、過去に投稿された直前の数件と一致するか、しないかで判別すればいいだけなので、ダブルポスト対策のルーチンを組み込むことは、さほど難しくはありません。

・タグを一定量以上使って投稿された記事を蹴る
→HTML のソースをモロ張りした記事では、タグの量が尋常ではありません。ですから、10 個以上タグを使った記事は投稿ができない、といった対策が有効かと思われます。

・記事の投稿可能なバイト数の上限を決めておく
→こうすることによって、クソ長い記事をウザウザと書かれる心配はありません。CGI プログラムのルーチンには、記事のバイト数を収得し、500k Byte 以上の量の記事は書き込みされない、といった対策が良いと思われます。

3、SSI コード書き込み攻撃

タグ攻撃を参照のこと。基本的にはタグさえ不許可にしていれば、まったく問題はない。

4、誹謗中傷・罵倒・個人情報ばらし攻撃

一番厄介、というか根本的な対策方法がないのが、この手の攻撃の特徴です。掲示板やチャットなどで、第三者を罵声で煽っているという光景は、もはや日常茶飯事だともいえますが、面識の無い者にいきなりこのようなことをされては、腹が立つのも無理ではないでしょう。そうやって、掲示板やチャット上で喧嘩を続けることによって、待ってましたといわんがごとく　「野次馬」が次々と参戦してきます。そうなると、その掲示板はもう戦乱のあと踏み荒らされた田畑と同じです。これには CGI 的な解決策はないのですが、あえてするといえば、特定単語のフィルター機能でしょう。これは特定の単語、例えば「バカ」「あほ」「氏ね」「ま●こ」などのような不謹慎単語が含まれていないかのチェックをして、もし入っていた場合、置換処理か書き込み拒否をするといった機能です。

5、Duke 攻撃

これは、他サーバーから、C や Perl などの言語を使用して書かれていて、設置された別サーバーから掲示板自動書き込みプログラムを全自動で動かし、永遠にどちらかのサーバーがダウンするまで書き込みを続けるという、迷惑きわまりないものです。また、この手のプログラムの厄介なところは、書き込みのフォーム内容を自動的に解析して、本当に人間が書き込みをしているかのように、メッセージをランダムに変えたりできるタイプの

ものもあるといことです。これには以下のような対策が無難でしょう。

・フォーム投稿時の暗号コードの同時投稿
→これによって、掲示板は書き込むそのときにリアルタイムにロードされ、それが書き込むために使用されたかということがわかります。ロボットプログラムによる自動書き込みは、大抵、以前にロードした投稿フォームを使いまわして投稿しているため、こういった策はかなり有効である。しかし、このような暗号コードを解析して、そのときに合った暗号コードを自ら作成して投稿する Duke プログラムも存在していることから、画像による暗号コードの人間の手による手移しで、投稿フォームの信憑性を高めようという仕組みの掲示板も最近は増えてきています。さすがに画像に表示された記号をロボットが認識するのは至難の技だからです。

ここまでで、さまざまな掲示板荒しの手口がおわかりいただけたと思いますが、これは表に見えているほんの一部の手法にすぎません。詳しい人ならいくらでも方法はあるものです。しかし、基本的な掲示板のセキュリティー対策（他ドメインからのフォームの書き込みをリーファーで判別して、禁止するとか、投稿メソッドに GET を使えないようにするとか、投稿者の接続 IP アドレスの表示など）さえしていれば、掲示板は相当荒らしにくいものです。ですから、あらされて困っているひとは、なぜ自分の掲示板がこのような目に遭うのかを、根本から考えた方がいいでしょう。掲示板、チャットにおける社会的秩序を守ることが、掲示板荒しを減らす一番の対策なのではないでしょうか。

以上

FreeBSD インストールに悪戦苦闘……

電子情報工学科　1回生　山本　大介

「まさかこんなに苦労することになるとは……」Windows 上ののソフト扱いはそこそこ慣れていた私ですが、それがコンピュータの知識のほんの一部であることがよ～くわかりました……。Windows の再インストールとはわけがちがう……。FreeBSD の入手法から日本語環境の構築までやることが多い、多い……。正直、日本語処理がコンピュータ上であれほどややこしいものとは思いませんでした。

●インストールにあたり……

コンピュータ部に入部したときは、UNIX というものを全く知らなかった私ですが、インターネットにおける UNIX 系 OS の重要性を知りました。そんなわけで箱の gaia でいろいろと「Hello World」を印字したりいろいろと試してみたわけです。最初のうちは DOS とのちがいがよく分からなかったので、「なんでいまさらコンソール画面で操作するんだ～」などと思ったりしてしまいました。それで、あまり FreeBSD の重要性をしらないまま、先輩に騙されたと思って FreeBSD のプログラミングなどをしてみることにしました。ところが、少しずつやっていくと、大学の Solaris や Irix 等の UNIX 系 OS のしくみが良く似ていることがわかり UNIX 系 OS の汎用性が高さを感じました。そして FreeBSD を勉強すればかなりコンピュータシステム勉強になると思い、FreeBSD を家のコンピュータに入れることを決断しました。

●導入前の問題

最初の問題はハードディスク容量でした。学科推薦のノートパソコンを持っていない私は家にある5年前のパソコンに FreeBSD を入れるしかありません。しかし、そのパソコンのハード容量は 610Mbyte+540Mbyte のせいぜい 1Gbyte ちょっと。Windows でそのほとんどを使用していたため、FreeBSD を入れる余裕はありませんでした。せめてもの救いは、PC-98 ではなく富士通の PC/AT 互換機だったということです。そこで、思い切って IDE の 15Gbyte ハードディスクを購入することにしました。ただ、マザーボードが古く 8Gbyte 超の領域が BIOS で見えないため、8Gbyte 超のハードディスクを入れた場合 Windows すら起動できなくなる可能性もあり、この購入はある意味賭けでした。購入後、早速ハードディスクを購入して、きちんと 8Gbyte だけ認識してくれることを確認して一安心……。早速、FAT16(Windows95 のため)で 2Gbyte(FAT16 の容量制限により)ずつスライスをつくり、さらに興味のある TRON の分のスライスをその後ろの 1Gbyte に確保し、7Gbyte 目以降を FreeBSD に割り当てることにしました。「これで 8Gbyte を超えないから boot の時も安心！」と思ったら、後に FreeBSD4.1 からは 8Gbyte 以上の領域からの

boot に対応していることがわかり、無駄な努力であることが判明しました……。

●インストール開始

そして、昔のハードディスクにはいっているデータを MO 経由で Windows のスライスに移動した後、FreeBSD のインストールを開始しました。インストール用の CD-R は先輩から譲ってもらい、早速 CD で Boot してインストール……。と思ったらなぜかうごかず……。TRON の時はすんなり CD で boot できたのですが原因不明のまま CD での boot するのをあきらめ、Boot 用フロッピーを作成しました。そして、もう一度インストール開始！今度はうまくいってるぞ……。順調に起動する FreeBSD。ところが「sio1:」の表示がでたまま突然停止……。CHAT で先輩に聞いてみたところ、シリアルポートに原因があると判明。早速、シリアルポートをインストールディスク起動時のデバイス設定メニューで認識しないようにしました。

●起動と終了

次に、スライスをぜいたくに 7Gbyte 目から最後までの分、8Gbyte 分とってパーティションの設定はよくわからないのでオートの設定のままの振り分けにしました。パッケージ等ははあまりよくわからなかったので ALL を選択し実行……。ゴリゴリ……ゴリゴリ……今考えるとかなり愚かだった……いつまでたっても終わらず……しばらく放っておくことにしました……。しばらく後インストールが完了し早速再起動。まずは普通のユーザでログイン……。しかし、ここで重大なことに気が付きました。「ルートユーザでログインするにはどうすればいいのだろう……。」今でこそ、root にすればすむと笑える話ですがそのときは真剣にわかりませんでした。かれこれ CHAT で聞いた後 root になって再度ログイン。いろいろコマンドを入れて動作を確認しました。ここで、再び問題が……。「終了の仕方がわからない！」大学の Solaris はログインネームとパスワードに shutdown を入力すれば終了し、BOX は halt01 です。この終了の仕方が特殊な方法だということを知らなかった私は、ログインネームにあらゆる名前を入れて終了させようとしました。当然、終了できるわけがなく、Ctrl+Alt+Delete で強制終了。あとで root で halt を入力すればいいことがわかり、終了は root しかできないものであることを知りました。

●X-Window の起動

このままでは永遠と苦労しないといけないので、BOX のような環境に近づけようと X-Window の KDE を立ち上げることにしました。ただ、そもそも X-Window システムがどういうものか知らないためとりあえず CHAT で聞いたとおり xinit を実行しました。すると、どこ

で設定したのか KDE が立ち上がり(この時は X-Window とはこういうものだと勘違いしてしまいましたが)メニューがあらわれたので端末エミュレータを起動しました。まず、mule とうってみましたが何にも起動せず。emacs でも動かず。xemacs と入力して初めて起動しました。が、まだ emacs の使い方をよく知らなかったので KDE 付属の当面はテキストエディタを利用することに……。なぜ Window マネージャが KDE だったのか確認するため、.xinitrc をみると exec startkde とかかれていて、ここで起動が設定されていると知りました。

●案外楽な PPP

次にインターネットに接続できないと話にならないということで PPP の設定を KDE でることに……。ところが、インストール時にシリアルポートの 1 と 2 を消した結果、どういう因果か PCI のモデムがそれに影響されて sio5:になったため 4 までしか選択項目のない KDE 標準の PPP ソフトは使用不能に……。先輩も「男ならコンソールで PPP やれ！」ということなので/etc/ppp/ppp.conf を FreeBSD ハンドブックを参考に設定。しかし、通常のプロバイダーの標準である PAP や CHAP の設定と UNIX ログインの設定の方法がちがうことが最初わからずがんばって UNIX 用の設定をしましたがうまくいかず AT 初期化コマンドとかいろいろためしました。そして PAP や CHAP の時の設定方法が思いっきり簡単なことがわかり脱力……。あっけなく接続は完了しました。

●なかなかうまくいかない……

早速パッケージにネットスケープが無かったため KDE 標準のチャットソフトを起動して irc.kitcc.org に接続。つながったものの、案の定文字化け。あとで聞いたところによると KDE は日本語に対応していないとか……メニューを日本語化する language っていう設定画面があるのに日本語に設定すると文字化けし、どうにもこうにもならない……。結局日本語化パッチは FreeBSD4.x には対応していないらしく、KDE 自体を諦めることにしました。また、KDE を諦めるのと前後して xinit 自体がきかなくなると言う事態が発生……。原因は.xinitrc に setenv 文を書いてしまったことのようでした。さらに、.login に exec xinit と書いていたのでログインと同時にログアウトするはめに……。シングルユーザモードで書き直せばよかったようなのですがよくわからなかったので、再インストールすることにしました。

●再インストール

つぎのインストールは前回の CD があまりにもパッケージが少なかったので UNIX User 購入して日本語パッケージの豊富な付属の CD の FreeBSD をいれることにしました。また、前回のインストールで必要なものがすこしずつわかってきたので、パッケージはいると思われるものだけをインストールする事にしました。あと、前回はパーティションの設定で root にあまり容量を割り振らなかったため、/root の容量がいっぱいになってしまって何もできなくなってしまうという悲惨なことになってしまいました。その反省をふまえ root ディレクトリは容量を多めにし、いろいろ作業ができるようにしました。また、パッケージとして mule や ja-netscape、wnn、kinput2 など日本語を使用するのに必要なものをそろえました。

●X-Window 変遷記

ところで、Window マネージャを決めるに当たり、Gnome や KDE はあまりにも X-Window の環境を改造しすぎていて X-Winodow がどういうものかわかりにくく、これから X-Window プログラミングを始めようと思っていたこともあり、シンプルな twm にする事に決定！KDE とちがいって Window マネージャが立ち上がってもメニューとかは起動しないので.xinitrc に kterm を書いたりして、シェルを立ち上げておかないと何もできなかったりします。ただ、もともと twm はコンソール指向なのでよけいなものがないほうが使いやすいという人向けですので、文句は言えませんが……。とくに 800×600 が最大画面のこのディスプレイ場合、余計なメニューのない方が使いやすかったというのもあります。

●qvwm の誘惑

そんなこんなで、twm でいこうと決心しかけたとき、悪魔の呼び声に耳を傾けてしまいました……。qvwm～。Windows ユーザにとってこれほど使いやすい環境はありませんでした。スタートメニューは便利！タスクバーは KDE より使いやすい！Window サイズの変更は楽々。デスクトップのショートカットもある！そして Alt+Tab による Window の切り替え！やはり Window マネージャはこうでなくては！なんてことをいったら、先輩から非難が続々と……。それでも今は qvwm で我が道を行っています。

●kinput2 に苦労する

ところで、qvwm を使い初めて kinput2 起動方法を知り早速 kinput2 -wnn & で起動しました。Shift+Space という起動方法にも驚きましたが、一番驚いたのは初期設定でスペースを押しても変換ができない！まさか Ctrl+C が変換だとは夢にも思いませんでした。

この設定はかなり使いにくい。この問題は先輩に ATOK 用のキー設定ファイルを頂いて、ひとまず決着。ところが、またもや問題が！Netscape で kiput2 を起動すると Window フォーカスの奪い合いになって、kinput2 の Window と Netscape の Window が点滅してしまうことに……。Netscape でも検索画面などのような Window の下にコンソール用のバーがついているものは大丈夫なのです。ところが、Netscape のメインの画面でフォーム等への入力すると点滅して入力が難しいものになってしまう……。そして、「これは qvwm のせいだ！」と思いそこらじゅうのホームページを探し回りました。そして、qvwm と kinput2 を同時に使うとそういうことが起こるので.qnwmrc に Kinput2 NOFOCUS と記述しないといけないことが判明。しかも、kinput2 ではなく Kinput2 と最初が大文字でないとだめだという……。とにもかくにも、これでなんとか Netscape で日本語が入力できるようになった……。でも、せっかく kinput2 が使えるようになったのに Netscape の場合 BackSpace すると 1byte ずつ消していくので 2byte 文字を BackSpace すると思いっきり文字化けしてしまったりする……。

●mule カスタマイズ

一方、mule では JSERVER localhost を設定して一足早く漢字変換ができるようになっていました。初めて変換できたときは感動……！日本語が使えるだけでこれほど感動してしまうとは……。また、Ctrl+¥という漢字キーはあまりにも使いにくいため[変換]キーを漢字キーに割り当てたり、[Del]キーで削除ができるようにしたりしました。irchat を mule にいれて FreeBSD 上からもチャットができることも知り、mule がいかに拡張性の高いソフトかを知りました。ただ、Ctrl+[を押したとき訳の分からない文字出力しかできなくなるのは、いまだに理解することができません……。mule は本当に奥が深いです……。

●パッケージ

話は飛びますが、ネットに繋げるようになりオンラインでパッケージをとりにいけるようになり、pkg_add とうつだけで簡単に、インストールできることを知りました。しかし、そのパッケージの使い方はどこにも書いてなかったりするので結局いろんなサイトをまわることになったりします……。とくに jman を入れたときは box（FreeBSD2.x.x）で LANG=ja_JP.EUCjp と.cshrc に記述していたのでそのまんまコピーしていたら jman が動かなかったりしました……。FreeBSD3.x 以降は LANGja_JP.EUC でないとダメなのだそうです。ポーツのインストールの使い方などは少し判ってきたのですがしくみが今ひとつしっくりきません。C の勉強して make のしくみをもっと詳しく知りたいと思います。他にもポーツに多い tar.gz 形式が tar が圧縮ソフトではなかったと知らなかったり、いろいろ FreeBSD の勉強になりました。

●これから

/usr /var /home /root /bin /sbin など各種ディレクトリの配分の仕方も少しずつ判ってきて FreeBSD がいかにディレクトリでファイルがきれいに分類分けされているかがわかりました。/usr や/home /var が別パーティションだったりする理由や、/usr のなかでバイナリが区分されている理由、はじめはややこしいだけで何もわかりませんでしたが、ちょっとずつわかってきた気がします。これからも、FreeBSD の勉強を続け C プログラミングともども将来役に立つよう身に付けていきたいと思います。