平成 14 年 11 月 20 日
京都工芸繊維大学コンピュータ部

Lime

はじめに

どうも、コンピュータ部の部長をしておりますコシモトという者です。今回も Lime を発行出来ることを嬉しく思い、まずは昨年に引き続き全編集をしてくれた山本君に感謝です。そしてこの冊子を印刷して下さった POPLS の方、並びに、これを今手に取って頂いている貴方に、部を代表して感謝します。

さて今回の Lime ですが、下回生の記事が少ないように感じられることと思います。もっと情熱をっ！と下回生に向けてビートを刻みたいです。しかしもしかすると、情熱を持てるようなものが無いのかもしれない、そんな想いに駆られることが最近あります。もっとも、僕らがそんな弱気ではいけないわけで、やはりもっとがんばらなくてはいけないんだと思います。

なんだか愚痴っぽい言葉になりましたが、これからの意気込みを胸に秘めて、はじめの言葉にさせて頂きます。でわまた、機会があれば、いずれどこかで。

平成 14 年 10 月 20 日
京都工芸繊維大学コンピュータ部部長 越本浩央

目 次

第I部

ハードウェア

1　Final Answer n+1

99220709 機械システム工学科 精密加工研究室 4 回生 奥谷 功一

はじめに

大学生を 4 年続けてこれが最後？の Lime の原稿となります。そこで今まで大学生活の中でふと疑問に思ったことを調べて解決してみました。なお昨年と同じように最初に定義をしておきたいと思います。文中にしばしば出てくるマスター CD またはプレス CD とは一般向けに市販されている CD のことを指します。業者の言うマスターアップのマスターとは異なりますのでご注意ください。名づけて “Final Answer” 早速いってみたいと思います。

1.1　CD-R の品質と焼きこみ速度の関係は？

まず CD や DVD の読込方式から説明します。

光ディスク上にレーザー光を照射し、乱反射する「ピット」と全反射する「ランド」の反射率違いによって読み取っています。ピットとランドには CD では 3T〜11T の 9 種類の長さがあり、実際の信号読み出しの際では反射率の変化を起きた地点を「1」、それ以外をすべて「0」として読み取ります。例えば、3T のピットやランドを持つデータは 3 ビット分のデータを示すことになります。

これを前提に書き込みはまったく逆のことを行っています。これらのデバイスでは、レーザー光のオン／オフを短時間に繰り返し、決められた長さのピットとランドを作ります。

もちろんレーザー光を使うわけなので照射のあとには「余熱」によって引き起こされる反応まで計算したシビアな照射の調整（ライトストラテジ）が必要になります。速度が速くなれば、その調整も容易ではありません。そのため速度が遅い方が安定しやすい構造になっています。さらに、ディスクが高回転になると、偏芯や風圧などによって引き起こされるブレや振動なので書き込みレーザー光の焦点がずれ信号の記録が正確に行えないといったことが起こりやすくなります。つまり、高速になればそれだけ品質を劣化させる可能性が増えることになります。

しかし、昨今の高速書き込み対応ドライブの急激な普及につれて CD-R 自体が低出力のレーザーで書き込めるように書き込む層が非常に薄くなってきています。なぜ高速書込みでは低出力になってしまうかというと、レーザーである面に焼きこむ力はレーザーの出力が一定なので時間に比例するからです。そこに、低速で書き込むと高出力のレーザーが当たることになり CD 自体がエラーを起こしてしまうようです。

後はメーカー自体の品質です。よく太陽誘電（以下誘電）が良いとか、TDK のタフネスが良いと

かいろんな事を聞きますが、これは本当に良いメディアなのでしょうか？

今までのことを考えれば簡単にわかります。どのメディアが良いメディアとは一概に言えません。たしかに誘電はレーザー出力に幅を利かせているため、大概のドライブできれいに焼けて、そして読むことが出来ます。しかし、本当の質というのはわからないみたいです。

わたしがよく見ている掲示板 http://www.muuz.ne.jp/cdr/にこのような情報がありました。これを書いてくれた人は CD-R の品質を検査する会社に勤めている人です。嘘か本当かはわかりませんが、適正な意見は述べていると思います。

>CDRW-drive 開発の仕事をしているものです。正直なところ台湾製 Media には苦労しています。Ritek & CMC は同じ lead in time でも全く色素の熱特性が違います。Princo は外周でのアシンメトリが気になるし … 日本企業でも三菱化学とか TDKってあんまりよくないです。ちなみに僕はいつも大事なものを焼くときは、太陽誘電のマスター用を使っています。これは C1C2error だけでなくアイパターンとか見たうえ一番信頼できるかなって思って使っています。やっぱ太陽誘電が一番です。最近品質落ちたけど、それでも一番。なによりレーザ Power のレンジが広いのがいいですよ。同じ製品でも drive によって差があるけど、太陽誘電ならかなり吸収してくれます。

わたしからの質問 1

太陽誘電のメディアのことなのですが、これは保存にも適しているのでしょうか？保存は三菱と聞いたことがあります。

A1

>これについては僕にはわからないです。実際だれもわからないのでは。意識しているのはやはりレーザ power ですが、先にも書きましたが、ほんとに強い power で焼くと寿命が短いのかどうか … 耐光材入っているとほんとに寿命が長いのかどうか …

確実に言えるのは時間に比例して劣化することです。で、焼き具合があまりよくなかったものは、最初ギリギリ読めていても、読めなくなるレベルになるのが早いです。そういう意味で太陽誘電は安定したところで焼け、またレンジも広いので悪くなっても読める可能性が高いと思います。TDK などは CD-CAT's で規格内にあっても、drive によって読めないことがありますが、逆に太陽誘電は規格外でも問題なく読めることが多いです。それはおそらく計測装置ではわかりにくい PIT/LAND のメリハリというかエッジの鋭さがあるんだと勝手に思っていますけど。そう考えると、適正なレーザがあたったときの三菱などアゾはたしかに保存性があるかもしれません。しかし極端に Power レンジがせまいので市販の drive では ??? です。

私からの質問 2

今の高速書き込み対応の R では低速書き込みするとうまく焼けない場合があると聞いたのですが、これは本当なのでしょうか？

A2

>高速書き込みになって色素構成が変わってしまうと低速書き込みが苦しいのはよくあることです。それはフタロによくあることですけどね。最近の高倍速フタロはほとんどその傾向があります。感光性が鈍いのか低速では power 強くしても規格内には収まらず、Jitter など悪くなる一方で … ついでにうち（書いてくれた人が勤めている場所）では、工場で TEST する

> とき、その Media での最高速度およびキャリブレーション倍速でしか TEST していません。低倍速については、開発の段階で数十台 check する程度です。他社さんも高速化に必死なので、低速はあまり検証していないのでは。
>
> つまり、書込みの最適速度はマシンそれぞれ CD-R 一枚一枚によって異なります。

1.2 音楽 CD を CD-R にバックアップを取った際、音質が落ちる？

音楽の話をしているときに、「CD-R に焼きこんだものよりもマスター（普通のプレス CD）のほうが音質が良い」ということを聞きました。しかしこれは本当なのでしょうか？

まず先ほど 1-1 で述べたことを思い出してください。次にジッタ（Jitter）というものの話をします。

ジッタとは読み出した信号の時間軸方向の揺れ（ズレ）のことです。エッジ（反射率が変動する地点）の不確定量ということもできます。ジッタが発生する要因としては記録されているピットやランドが本来あるべき長さ（基準の長さ）よりも短かったり長かったりと物理的にばらつく「デビエーション」と呼ばれる現象があります。ピットやランドは理論上はばらつきがないはずですが、レーザー光を利用して書き込みを行う CD-R などは熱によりレーザーの波長が伸び長さがばらつくことがあります。またメディア面のブレや偏芯などによってもレーザー出力が最適値から外れ、同様の現象が起きる可能性があります。なによりマスター CD などでもスタンパ状態が劣化してくるとデビエーションが発生することが少なくありません。また読み出し（再生）に必要な反射率などの機械特性が満たされていなかったケースも同様です。ジッタとは、つまり本来あるべき場所にエッジの位置を得ることができないため、クロックが時間軸方向に揺れてしまうことです。ジッタがひどくなると「0」と「1」の判別を間違い、エラーレートの劣化を招くことがります。

さらに、偏芯も問題です。1-1 で述べたことは書込みの際だけの話ではありません。読み込みの際も注意が必要です。以上のことより、基本的にはマスターの方が良いと言った程度です。この音の差は人によっては聞き分けられるみたいなのですが、それは神の領域といっても過言ではないです。

> そこそこ良いメーカーの CD-R を使っている限り差はないといえる。

というのが私の持論です。

ちなみに軸ぶれなどやジッタなどの誤差がマスターより CD-R の方が厳しいためマスター以上の音質を持った CD-R を作ることも可能です。かなり難しいとは思いますが…。

1.3 63 分ディスクは 74 分や 80 分ディスクに比べて質がいい？

一般的にエラーレートが低い、ジッタが抑えられているなど、機械特性が優れたものの方が質がいいのは明らかです。機械特性に優れた CD を作成するためのポイントは

- 品質の良いメディアを使うこと
- 偏芯や反りが少ないメディアであること

- 物理的に悪い部分がなるべく少ないメディアを選択すること、が挙げられます。
- そして物理的にマージンが最も高い 63 分メディアを使用するということも挙げられます。

74 分や 80 分のディスクと 63 分のディスクは線速度に違いがあります。線速度とは、単位時間あたりにピックアップが移動する距離のことです。線速度は 74 分や 80 分メディアで 1.2m/sec、63 分では 1.4m/sec となっています。これによりジッタの劣化が少なくなります。だから質の良い物ができるというわけです。

余談 1 CD の規格で線速度は 1.2m から 1.4m と決められています。90 分や 99 分のメディアはその線速度をもっと短くしたメディアであるため規格外というわけです。というわけで基本的には専用ドライブでないと読み書きはできないわけです。

余談 2 63 分の CD-R を手に入れるのが難しい場合は、YAMAHA の AudioMASTER 機能搭載の CD-R ドライブというものがあります（5 参照）。こちらの場合は 74 分や 80 分の CD-R に強制的に 1.4m/sec の線速度での書込みを行い、63 分にしてしまうという機能がついております。

1.4 音楽を CD-R にコピーすることはどこまで許される？

昔は友達から借りてきた CD をカセットテープや MD に録音してたとおもいます。これは多めに見られえていたのかは知りませんが著作権法違反ではないようです。しかし、WinMX 等で落とした音楽を CD-R に焼くことは著作権法違反です。いったいどこまでが著作権法違反ではないかということを調べてみました。

まず CD をレンタルする場合、レンタル業者は CD の使用料というものを CD リリース業者（avex 等）に払わなければなりません。これが著作権の中にある貸与権です。これはもちろん CD レンタル代に含まれているとおもいます。このことは我々が CD を借りる時点で著作権料（補償金）を払っていることを意味しています。つまり、あとは著作権法上の「個人利用の範囲内においては OK」ということになっていますので、レンタル CD を CD-R に焼いても違法ではありません。しかし、その CD-R を他人に販売したり譲渡するのは、個人利用の範囲を超えていますので違法となります。

ちなみにレンタル屋さんがこの料金を払っているかどうかはレンタル CD に日本音楽著作権協会（JASRAC) の印（一般にはシールの場合が多い）の有無です。詳しくはこちら http://www.jasrac.or.jp/info/rental/

さらに、知人から CD を借りて、CD-R にコピーする場合「音楽 CD の場合は、家族や友人で自由に貸し借りし、テープやミニディスクに録音するのは自由」とされています。六法を見ると、著作権法第 30 条で営利目的でない私的な目的なら著作物のコピーが認められていますので OK です。これはあくまで著作権法上ではなく「私的使用であるために複製しても問題がない」という意味です。

また、現在のところ CD-R での複製製作は政令（著作権法施行令第一条）で定めるデジタル機器（とメディア）に該当しないため、法律上は補償金を著作権者に支払う必要はありません。

これの延長で WinMX でやりあっているのはわたしのネット上の友人だという意見もあるかもしれませんが、オンライン上にアップすることは不特定多数のものに配布されるということになり違法という事になります。

しかも CD をみてみると「ネットワークを通じてこの CD に収録された音を送信できる状態にすることは、著作権法で禁じられています」とはっきり書いてあります。WinMX は絶対に違反ということになります。

以上のことをまとめると、境界線は

- 知人から借りた CD をコピーすることはセーフ
- そのコピーした CD を知人に貸すのはアウトといったあたりだと思います。

しかし、これは一弁護士の勝手な判断をホームページに載せている人がいましたのでそれを一部参考にさせていただいて書いたに過ぎません。特に友人から借りた CD をコピーすることに関しては、友人がオリジナルの CD を手放した時はその CD-R を手放さなければならないことなどがあるので、友人から借りた CD を CD-R 等にコピーすることは細心の注意が必要です。

1.5 (Advanced)AudioMASTER 機能とは？

先ほど 1-3 で少し説明した AudioMASTER について詳しく説明します。

YAMAHA の CRW3200 シリーズや CRW-F1 シリーズが搭載した機能で、CD の規格内でもっとも高速な線速度を使用して記録することによって、音質の向上を狙う機能です。

CD で使用される線速度は「1.2〜1.4m/sec」と規定されており、読み出しや書込みでは、この範囲で常に一定の線速度が使用されています。

この機能は CD の規格の線速度のマージンを利用し、強制的に 1.4m/sec で書込みを行っています。信号を線の長さで記録している CD では、その原理上、書込みに使用する線速度は高速の方が安定した読み出しを行うことができます。特に、ジッタの低減には、最も効果的です。この機能を使えば、550MB しか書込みはできませんが、質の良い CD-R を作ることができます。さらに AdvancedAudioMASTER に進化した CRW-F1 シリーズでは 870MB（99 分）の CD-R もサポートし、79 分まで書き込みができるようになった上、今まで 4 倍しか対応していなかったこの機能に等速焼きと 8 倍焼きをサポートしました。（但し、Audio データ以外で使えるかどうかは不明）

1.6 IDE インターフェースは SCSI よりも性能が劣る？

かつて昔は SCSI といえばディスクインターフェースの代名詞であり実際にデータ転送速度も高速でした。しかし IDE も PIO 転送[1]から DMA 転送への進化、U-ATA133 などのデータ転送速度の向上により、SCSI に見劣りしなくなってます。むしろ SCSI の方が見劣りをしています。本当に接続台数といった物理的制限を除けば IDE は SCSI に性能的に見劣りしないのでしょうか？少しまとめたいと思います。

[1]PIO 転送とは ProgramI/O の意味で、プログラムからの要求によって CPU が直接デバイスとデータをやり取りを行う転送方式です。それに対し DMA 転送とは DirectMemoryAccess のことでその名のとおり CPU を介さずに、DMA コントローラを使用してデバイスとシステムメモリ間で直接データをやり取りを行う転送方式です。DMA コントローラを持っていない IDE デバイスは PIO モードでしか転送できないため IDE は規格上必ず PIO モードをサポートしていなければなりません。

IDE では原則としてデバイスがコマンド実行中はそのデバイスがバスを占有し、バスの調停機能もありません。SCSI ではより多くのデバイスが接続できるという都合上の問題もありますが、デバイスがコマンド実行中にバスを開放してほかのデバイスがバスを使用することもでき、バスの調停機能ももちバスを効率よく利用できます。

以上のことよりバスの効率を考えると SCSI に軍配が上がります。

IDE では CD-ROM 系のドライブと HDD 系のドライブを別系統のバスに接続した方が良いというのもこの関係です。

次に SCSI の HDD は IDE のものに比べ高速のものが多いです。そのためシーク時間が短くなっています。

この点も SCSI のほうに軍配が上がります。

もっとも転送速度が向上すれば、バス占有時間も短くなるため IDE と SCSI というインタフェースの違いは体感することはそう多くありません。ただし面密度が多くなっても回転数が一定の場合シーク時間が短くなることはありません。

しかし、SCSI の最大の欠点は値段の問題です。いまや 1GB あたり 100 円を切ろうとしている IDE の HDD ですが SCSI の HDD はいまだに 1GB あたり約 1000 円と非常に高価なものになっています。しかもこれは HDD という一例であり、ほとんどのものが SCSI では値段が高くなっています。

コスト的な問題では IDE に軍配が上がってします。

以上が公正な判断です。あとの IDE の性能がよいとか SCSI の方がよいというのは個人が決める問題です。わたしは SCSI 派の人間なのでこれ以上公正に書くことができません。

わたしが言いたいことは IDE 派の皆さん SCSI はまだまだ捨てたものではありませんよ。ということだけです。

1.7 Serial-ATAって何？また今までの機器との互換性は？

HDD など今までパラレルであった ATA がシリアルになるのです。

執筆時ではまだ Serial-ATA の HDD や CD-ROM などのドライブは製品化されていませんがおそらくこの本が発行するころには製品化されている規格です。

これによりケーブルが U-ATA133 では GND 含め 80 本だったのが、S-ATA では 7 本と細くなります。HDD 本体はシリアルで書込みを行っているので制御の簡略化によるエラーの減少などができると思います。

U-ATA133 の HDD とは現在は変換コネクタを利用して接続します。製品を持っていないので性能に関することはなんとも言えません。

ちなみに S-ATA 転送速度はとりあえず最初は 1200Mbps（150MB/s）です。現在の最高といわれている U-ATA133 よりも高速です。来年の頭くらいには本格的に HDD やマザーボードが順じ発売していくようです。

1.8 Last Fainal Anser（あとがき）

思い起こせば 3 年前、何を書いていいか分からずさらに、専門的な知識も無く阪急電鉄のことをひたすら書いた気がします。そして、去年 PC アーキテクチャについて少し書いて、自分でも力がついたなと実感しました。そして今年、あまり書いている時間が無く、結局昨年度から少しずつ書いてきた（本来 webLime に公開しようと思って細々書いてきたのですが…）この内容を提出することになりました。内容には少し古い部分が含まれているかもしれませんがご了承ください。

この大学生活で、かなり気になっていたことをまとめたつもりです。他にも SafeDisc のとこなどいろいろ調べたことはあるのですが、SafeDisc のことはこのようなところに載せる内容ではないと思いあえて掲載をしませんでした。他に調べたことも同様な理由です。　まぁ来年も機会があれば投稿させていただきたいと思っております。そして冗談は抜きにして HDD の事を書きたいと思います。

そして以上で Falnal Answer を終わりたいと思います。これを読んでくれた人に感謝を込めて…。

2002 年 9 月吉日

2　インクジェットプリンタの構造

02230055 電子情報工学科 1 回生 高篠 豪

2.1　はじめに

PC を利用している人ならば、殆どがお世話になっているプリンタ。その中でも現在主流と思われるインクジェット方式のプリンタについて少々調べてみました。

内容、文章共にまだまだ稚拙ですが、ご容赦願います。

2.2　インクジェットプリンタとは

ink-jet printer という名前の通り、インクを紙に噴射して印字するタイプのプリンタの事を指します。

最近主流のこの方式。何故これだけ普及した要因としては、何よりもプリンタ本体の値段が安いという点が大きいでしょう。他にも長所としては、印字音が比較的静か、カラー印刷が容易、等という特徴があり、逆に短所としては印刷速度が遅い、きれいに印刷しようとすると専用の用紙が必要になる、という特徴があります。もっとも、一般家庭では大量に印刷したり、格別きれいに印刷する必要性があまり無い為これらの短所が存在しても普及したのでしょう。

2.3　インクジェットプリンタの中心部、ヘッド

インクジェットプリンタは先程述べた様な特徴を持っているのですが、ヘッドと呼ばれる紙にインクを吹き付ける機能を持った部分の方式の違いにより、さらに特徴が分かれます。

ヘッドはインクカートリッジからインクを受け取り貯める部分 (インク室)、インクを噴出す為インク室から伸びた管 (ノズル)、インク室のインクを押し出す為の”何か”の三つで構成されています。

分かり難い方は注射器を思い浮かべてください。注射器の薬液を貯める部分がインク室、針がノズル、ピストンがインクを押し出す為の部分だと考えると分かり易いかと思います。

そして最後のインクを押し出す部分方式の違いがすなわちヘッドの機能の違いとしてあらわれ、各会社毎のインクジェットプリンタを特徴付けています。

以下、現在主流となっているエプソンのマイクロピエゾ方式、ヒューレットパッカードのサーマルインクジェット (TIJ) 方式、キャノンのバブルジェット (BJ) 方式の紹介をします。

2.4 マイクロピエゾ方式 (エプソン)

ピエゾ方式とは、電圧に反応して変形するピエゾ素子の圧力を利用することによってインクを噴射する方式のことです。

インク室内のノズルとは反対側にピエゾ素子を配置し、電圧をかけることによりピエゾ素子がノズル側に押し出される様に変形させればノズル先端のインク液面が盛り上がり、その瞬間ピエゾ素子を引っ込めればノズル面よりも前にあるインクが千切れ、慣性により噴射されるという方法で印刷します。

ピエゾ方式のメリットはこの素子にかける電圧を制御することでインクの押し出しを細かく制御出来る、という点にあります。その為大きなノズルを用いても小さなインクを噴射することができ、単一のノズルで複数のドットサイズのインクを噴射することが可能になります。その反面、ノズル数を他方式よりも増やすのが難しい、という欠点も持ち合わせています。なぜならば構造的に小型化が容易ではない上に、比較的 1 ノズル毎のコストが高くなる為です。

他にも、インク室が大きくなってしまう為、内部に空気が侵入したときに排出し難いという難点もあります。水蒸気爆発を利用するサーマル方式に比べ、ピエゾ素子では空気を押し出す為の圧力が足りない事がその理由です。

2.5 サーマルインクジェット (TIJ) 方式 (ヒューレットパッカード)

この方式ではピエゾ方式と違い、ヒーターを利用してインクを噴射する方式です。

インク室内のノズルとは反対側にヒーターを配置してインクを加熱、インク室内に出来る気泡がはじける力によりインクが噴射される、という仕組みです。

TIJ 方式最大のメリットは非常に単純なヘッド構造にあります。ピエゾ方式のそれと違い、ヒーターをプリントした板の上にインク室とノズルと取り付けるだけなので、高精細化と大量生産で有利と言えます。その為今後ともノズル数を増大させることが容易と言われています。また、ピエゾ方式とは逆にインク室がコンパクトで、気泡の弾ける力が大きい為、内部に空気が侵入してもインクと共に放出され易くインク詰まりし難いというメリットもあります。

デメリットは単一ノズルで単一量のインクしか噴射出来ない為、小さいドットで印刷する場合印刷速度が低下してしまうという点です。ただし現在のところはそれ以上にノズル数が増加している為、デメリットとはなっていません。

2.6 バブルジェット (BJ) 方式 (キャノン)

この方式では TIJ 方式同様、ヒーターを利用してインクを噴射します。

それでは何が違うのかというと、大きな違いはヒーターの位置にあります。TIJ 方式の場合ノズルとは反対側にヒーターを設置しているのですが、BJ 方式の場合はインク室側面にヒーターを設置しています。

インク室側面から加熱して気泡を作り出して、インク室いっぱいになるとその圧力によりインク

が噴射されます。

この方式では上記の二つの方式と違い、ピエゾ素子や気泡によるプッシュープル動作 (押し出して引きちぎることで吐出する動作) ではなく、インクを気泡で分断して、そのままノズル側のインクを吐き出すという手法でインクを噴射しているため、エネルギーロスが少ない高速のインク噴射を可能にしています。

何故高速でインクを噴射した方が良いのでしょうか？高画質の画像をきれいに印刷するならば、個々のインク滴のサイズを小さくし、細かくインクを噴射する必要があります。インク適が小さくなると一滴当たりの重量が軽くなり、その分インクをまっすぐ飛ばすのが難しくなります。そしてインク滴の重量を増やす、つまりインク滴を大きく出来ない以上きれいにインクを飛ばすにはインクの飛行速度を高め、インクの持つエネルギーを高める必要があるからです。

また、あらかじめ気泡でインクを分離してしまう為、インク滴のサイズが安定するというメリットもあります。インクを同じサイズのドットで揃え易く、きれいな印刷が可能になります。

2.7 おまけ話

ヘッドの説明の際、注射器を利用して説明しましたが、実はバブルジェットプリンタはその注射器をヒントに開発されたと言われています。事の起こりは 20 年以上前、とあるメーカーがプリンタの研究をしていて、そこでは細かい部品にインクを注入する為に細い針を持つ注射器を使っていました。ある日、研究員の一人がハンダゴテ (ハンダ付けに使う発熱する工具) をうっかり注射器の上においてしまいました。加熱されていたハンダゴテの先が注射針に触れると、その瞬間針の中のインクが勢い良く飛び出しました。その現象にヒントを得て、研究員達は新たなヘッドの開発にのりだしたそうです。その後、インクを加熱することによるノズル詰まりの様な問題を解消して、現在の形のヘッドを持つプリンタが開発されてきました。

第II部

ネットワーク

1 FreeBSD はこれくらい使ったらんともったいない

99230016 電子情報工学科 パターン情報処理研究室 4 回生 大宮 広義

こんにちは。僕は現在電子情報工学科 4 回生をやっています。パターン情報処理研究室に配属されていて、研究をする毎日です。最近コンピュータ部の活動には、めっきり関われないでいますが、今回はひとつ実用的なことを書いてみました。「コンピュータ部で FreeBSD いじってるんやったら、ルータとかチャッチャと作れてほしい!」そんな切なる願いから、少しは後進に貢献できればと思います。

さて、最近ブロードバンド環境は至極当たり前のものになりつつありますね。ADSL モデム付属の機能だけで満足してませんか? そんでもって、ADSL のモデムの内側の FreeBSD で WEB サーバを公開したりして、せっかく立ち上げっぱなしの FreeBSD マシンを粗末にしていませんか? FreeBSD はサーバにするには最高クラスの OS です。そんな素晴らしい OS を有効に使ってやらないと、もったいないお化けがでます。

先ほど FreeBSD でルータを作れてほしいといいましたが、正味ルータだけではもったいないのは当たり前。というわけで、「使える」FreeBSD を作っちゃいましょう。実際、この「使える」FreeBSD を作るのは、さほど時間のかかることではありません。要するに、要領さえ得てしまえば、チャッチャと作れるようになるのです。なら、身につけておかない理由はないでしょう。

さて、下の方にコマンドの無機質な羅列が見えますね。基本的に、全部打っていけば出来上がるようにしてあります。エディタなどで編集すればいいところでも、

```
# cat > file
```

でわざわざ書いてありますが、vi でも emacs でも気にせずに自分の好みのエディタを使ってください。その方が安全です。では、がんばって行きましょう。

まず、ソースツリー、ports ツリーを含めた FreeBSD をインストールした PC を用意してください。それには、2 枚の NIC が刺さっている必要もあります。今回の例では、外側のインタフェースが xl0、内側のインタフェースが xl1 となっています。つまり、xl0 が ADSL モデム側、xl1 がその他のクライアント側となります。また、ADSL モデムの IP アドレスは 192.168.0.1 とします。あと

ブロードバンド環境はモチロン必要ですよっ。これだけあれば準備万端。login して root になります。後はひたすらコマンドを打っていくのみです。

```
[まずは外側につなげないと意味がない]
# hostname gw.otsu-city.com                                    (ホストネームを設定)
gw# ifconfig xl0 inet 192.168.0.254 netmask 255.255.255.0     (xl0 の IP アドレスの設定)
gw# ifconfig xl1 inet 10.0.0.1 netmask 255.255.255.0          (xl1 の IP アドレスの設定)
gw# route add default 192.168.0.1                             (経路の設定)
gw# echo 'hostname="gw.otsu-city.com" >> /etc/rc.conf         (以下、起動時に設定されるようにする)
gw# echo 'ifconfig_xl0="inet 192.168.0.254 netmask 255.255.255.0"' >> /etc/rc.conf
gw# echo 'ifconfig_xl1="inet 10.0.0.1 netmask 255.255.255.0"' >> /etc/rc.conf
gw# echo 'defaultrouter="192.168.0.1" >> /etc/rc.conf

gw# echo '192.168.0.254 localhost gw gw.otsu-city.com' >> /etc/hosts
gw# echo 'nameserver 61.199.217.76' > /etc/resolv.conf        (名前の解決)
gw# echo 'domain otsu-city.com' >> /etc/resolv.conf           (ドメインの設定)

gw# sysctl net.inet.ip.forwarding=1 >/dev/null                (NIC 間の通信を有効にする)
gw# echo 'gateway_enable="YES"' >> /etc/rc.conf               (起動時に有効に)

gw# kldload ipl                                               (ipfilter のために kernel module を
gw# kldstat                                                    有効にする。kldstat で確認)
Id Refs Address    Size     Name
 1    3 0xc0100000 3ce1f0   kernel
 2    1 0xc2afa000 18000    ipl.ko

gw# echo 'map xl0 10.0.0.0/24 -> 192.168.0.254/32' > /etc/ipnat.rules (ipnat のルールファイル)
gw# ipnat -CF -f /etc/ipnat.rules                             (ipnat のルールを適用)
gw# echo 'ipnat_enable="YES"' >> /etc/rc.conf                 (起動時に有効に)
gw# echo 'ipnat_rules="/etc/ipnat.rules"' >> /etc/rc.conf

[IPFilter の設定]
gw# perl /usr/src/contrib/ipfilter/mkfilters | grep -v inet6 > /etc/ipf.rules (ルールファイル)
gw# ipf -Fa -f /etc/ipf.rules                                 (ipfilter のルールを適用)
gw# ipmon -D /var/log/ipf.log                                 (ipmon を有効に)
gw# ipfstat -i                                                (入ってくるパケットのルールを確認)
gw# ipfstat -o                                                (出て行くパケットのルールを確認)
gw# echo 'ipfilter_enable="YES"' >> /etc/rc.conf
gw# echo 'ipfilter_rules="/etc/ipf.rules"' >> /etc/rc.conf
gw# echo 'ipmon_enable="YES"' >> /etc/rc.conf
gw# echo 'ipmon_flags="-D /var/log/ipf.log"' >> /etc/rc.conf
```

ルールファイルは、Google でサンプルを調べるなり、man を読むなりして、もっと自分のセキュリティポリシーにあったルールにしていきましょう。

```
[SSH サーバの設定]
gw# ssh-keygen -t rsa1 -N "" -f /etc/ssh/ssh_host_key         (ホストキーの作成)
gw# ssh-keygen -t rsa -N "" -f /etc/ssh/ssh_host_rsa_key
gw# ssh-keygen -t dsa -N "" -f /etc/ssh/ssh_host_dsa_key
gw# echo 'sshd_enable="YES"' >> /etc/rc.conf

[DHCP サーバの設定]
gw# cd /usr/ports/net/isc-dhcp3                               (dhcpd を ports でインストール)
gw# make install clean
gw# rehash
gw# cat > /usr/local/etc/dhcpd.conf                           (dhcpd の設定ファイル)
server-identifier 10.0.0.1;
option domain-name "otsu-city.com";
option domain-name-servers 61.199.217.76;
subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.100 10.0.0.110;
    option routers 10.0.0.1;
    option subnet-mask 255.255.255.0;
    option broadcast-address 10.0.0.255;
    default-lease-time 600;
    max-lease-time 7200;
}
(Ctrl+D)
gw# /usr/local/sbin/dhcpd xl1                                 (dhcpd の起動)
gw# cat > /usr/local/etc/rc.d/dhcpd.sh                        (起動スクリプトを作成)
#!/bin/sh
```

```
dhcpd="/usr/local/sbin/dhcpd"
interface="xl1"

if [ -x ${dhcpd} ]; then
    ${dhcpd} ${interface}
    echo -n ’ dhcpd’
fi

exit 0
gw# chmod 774 /usr/local/sbin/dhcpd.sh                    (起動スクリプトに実行権を)

[ネームサーバの設定]
gw# cd /usr/ports/ftp/wget                          (アーカイブのダウンロードには wget が便利)
gw# make install clean
gw# rehash
gw# mkdir /usr/local/share/src                      (作業用ディレクトリの作成)
gw# cd /usr/local/share/src
gw# wget ftp://ftp.isc.org/isc/bind9/9.2.1/bind-9.2.1.tar.gz
gw# tar zxvf bind-9.2.1.tar.gz                      (アーカイブを展開)
gw# cd bind-9.2.1
gw# mkdir /usr/local/pkgs                           (各種プログラムのインストール先を一括管理する)
gw# ./configure -prefix=/usr/local/pkgs/bind-9.2.1  (インストール先を指定して configure)
gw# make && make install
gw# ln -s /usr/local/pkgs/bind-9.2.1 /usr/local/pkgs/bind   (バージョン管理を楽にするコツ)
gw# echo ’named_enable="YES"’ >> /etc/rc.conf
gw# echo ’named_program="/usr/local/bin/bind/bin/named"’ >> /etc/rc.conf
gw# echo ’named_flags="-u bind"’ >> /etc/rc.conf
gw# cd /etc/namedb                                  (ネームサーバの設定ファイル置き場)
gw# cat > named.conf                                (ネームサーバの設定ファイル)
options {
    directory "/etc/namedb";
    auth-nxdomain no;
    pid-file "/var/run/bind/named.pid";
};

zone "." {
        type hint;
        file "named.root";
};

zone "localhost" {
        type master;
        file "db.localhost";
};

zone "0.0.127.IN-ADDR.ARPA" {
        type master;
        file "db.127.0.0";
};

zone "otsu-city.com" {
        type master;
        file "db.otsu-city";
}

zone "0.0.10.in-addr.arpa" {
        type master;
        file "db.10.0.0";
}
(Ctrl+D)
gw# mkdir /var/run/bind                                   (pid ファイルの格納場所)
gw# chown bind /var/run/bind                              (bind ユーザで実行するための調整)
gw# cat > /etc/namedb/db.localhost                        (以下、各種レコードファイルの記述)
$TTL 14400
@  IN SOA  localhost.   root.localhost. (
                        2001042601      ; Serial
                        86400           ; Reflesh
                        3600            ; Retry
                        3600000         ; Expire
                        3600            ; Minimum
                )
```

```
                IN      NS      localhost.

                IN      A       127.0.0.1
(Ctrl+D)
gw# cat > /etc/namedb/db.127.0.0
$TTL 14400
@               IN SOA localhost.  root.localhost. (
                        2001042601      ; Serial
                        86400           ; Reflesh
                        3600            ; Retry
                        3600000         ; Expire
                        3600            ; Minimum
                )

                IN      NS      localhost.
1               IN      PTR     localhost.
(Ctrl+D)
gw# cat > /etc/namedb/db.otsu-city
$TTL 14400
@  IN SOA  ns.otsu-city.com.   root.otsu-city.com. (
                        2002100601      ; Serial
                        86400           ; Reflesh
                        3600            ; Retry
                        3600000         ; Expire
                        3600            ; Minimum
                )
                IN      NS      ns
                IN      MX      1 mail
                IN      A       10.0.0.1

ns IN A 10.0.0.1
mail IN A 10.0.0.1
www IN A ns
gw IN A ns
(Ctrl+D)
gw# cat > /etc/namedb/db.10.0.0
$TTL 14400
@       IN SOA  ns.otsu-city.com.   root.otsu-city.com. (
                        2001093001      ; Serial
                        86400           ; Reflesh
                        3600            ; Retry
                        3600000         ; Expire
                        3600            ; Minimum
                )
                IN      NS      ns

1 IN PTR ns
(Ctrl+D)
gw# /usr/local/pkgs/bind/bin/named -u bind                  (ネームサーバの起動)
```

ここまでの設定で、この FreeBSD マシンはネームサーバも兼ねるようになったので、この FreeBSD マシンでローカルの PC や外部のホストの名前を解決できるようにする。

```
/etc/dhcpd.conf:
< option domain-name-servers 61.199.217.76;
> option domain-name-servers 10.0.0.1;

[メールサーバ (Postfix) の設定]
gw# cd /usr/ports/mail/postfix                          (Postfix のインストール)
gw# make install clean
(Choose [Cancel])
gw# rehash
gw# cat /usr/local/etc/postfix/main.cf                  (Postfix の設定ファイル。
queue_directory = /var/spool/postfix                    一から書かないで、デフォルトのをいじればよい)
command_directory = /usr/local/sbin
daemon_directory = /usr/local/libexec/postfix
mail_owner = postfix
myhostname = mail.otsu-city.com
mydomain = otsu-city.com
myorigin = $mydomain
inet_interfaces = all
mydestination = $myhostname, localhost.$mydomain $mydomain
mynetworks_style = subnet
```

```
mynetworks = 192.168.0.0/24, 10.0.0.0/24, 127.0.0.0/8
relay_domains = $mydestination
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
mail_spool_directory = /var/mail
smtpd_banner = $myhostname ESMTP $mail_name
debug_peer_level = 2
debugger_command =
         PATH=/usr/bin:/usr/X11R6/bin
         xxgdb $daemon_directory/$process_name $process_id & sleep 5
sendmail_path = /usr/local/sbin/sendmail
newaliases_path = /usr/local/bin/newaliases
mailq_path = /usr/local/bin/mailq
setgid_group = maildrop
manpage_directory = /usr/local/man
sample_directory = /usr/local/etc/postfix
readme_directory = no
maximal_queue_lifetime = 5d
gw# postfix start                                          (Postfix の起動)

[HTTP サーバ (Apache) の設定]
gw# cd /usr/local/share/src
gw# wget http://www.apache.org/dist/httpd/apache_1.3.27.tar.gz
gw# tar zxvf apache_1.3.27.tar.gz                          (アーカイブの展開)
gw# cd apache_1.3.27
gw# ./configure --prefix=/usr/local/pkgs/apache_1.3.27 --enable-shared=max (DSO を有効に)
gw# make install
gw# ln -s /usr/local/pkgs/apache_1.3.27 /usr/local/pkgs/apache
gw# cat >  /usr/local/etc/rc.d/apache.sh                   (起動スクリプトを作成)
#!/bin/sh

apache_home="/usr/local/pkgs/apache"

if [ -x ${apache_home}/bin/apachectl ]; then
        (cd ${apache_home} && ./apachectl start)
fi

exit 0
(Ctrl+D)
gw# chmod 774 /usr/local/etc/rc.d/apache.sh                (実行権を設定)
gw# cd /usr/local/pkgs/apache/bin
gw# ./apachectl start                                      (Apache を起動)
gw# telnet localhost 80                                    (起動していることの確認)
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.                                  (Ctrl+] でコマンドモードに)
^]
telnet> quit
Connection closed.

[OpenSSL のインストール]
gw# cd /usr/local/share/src
gw# wget http://www.openssl.org/source/openssl-0.9.6g.tar.gz
gw# tar zxvf openssl-0.9.6g.tar.gz                         (アーカイブの展開)
gw# cd openssl-0.9.6g
gw# ./config --prefix=/usr/local/pkgs/openssl-0.9.6g       (インストール先を指定して configure)
gw# make
gw# make test                                              (一応ちゃんと test しておく)
gw# make install
gw# ln -s /usr/local/pkgs/openssl-0.9.6g /usr/local/pkgs/openssl
gw# mv /usr/bin/openssl /usr/bin/openssl.OLD               (初めからある openssl を退避)
gw# ln -s /usr/local/pkgs/openssl/bin/openssl /usr/bin/openssl (symlink を張っておく)

[POP サーバ with OpenSSL の設定]
gw# cd /usr/local/share/src
gw# wget http://www.ring.gr.jp/archives/net/mail/qpopper/qpopper4.0.4.tar.gz
gw# tar zxvf qpopper4.0.4.tar.gz                           (アーカイブの展開)
gw# cd qpopper4.0.4
gw# ./configure --prefix=/usr/local/pkgs/qpopper4.0.4 --with-openssl=/usr/local/pkgs/openssl
gw# make && make install
gw# ln -s /usr/local/pkgs/qpopper4.0.4 /usr/local/pkgs/qpopper
gw# echo 'pop3 stream tcp nowait root /usr/local/pkgs/qpopper/sbin/popper popper' \
```

```
    >> /etc/inetd.conf                                    (inetd から呼ばれるようにする)
gw# ps ax | grep inetd
   84  ??  Is     0:04.41 /usr/sbin/inetd -wW
gw# kill -HUP 84                                         (inetd に設定ファイルを読み直させる)
gw# telnet localhost 110                                 (ちゃんと動いているか確認)
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK Qpopper (version 4.0.4) at localhost starting.
^]
telnet> quit
Connection closed.

[Qpopper 用証明書の作成 (おまけ)]
gw# cp /usr/local/pkgs/openssl/ssl/misc/CA.sh /etc/ssl   (証明書作成スクリプトをコピー)
gw# cd /etc/ssl                                          (SSL の設定ファイルの置き場)
gw# chmod u+x CA.sh                                      (実行権を設定する)
gw# ./CA.sh -newca                                       (CA の秘密鍵と自己署名された証明書を作成)
(必要な情報を入れる)
gw# ./CA.sh -newreq                                      (ユーザの秘密鍵と公開鍵を作成し、CSR を作成)
(必要な情報を入れる)
gw# ./CA.sh -sign                                        (CA の秘密鍵を用いて CSR に署名し証明書発行)
(必要な情報を入れる)
gw# openssl rsa -in newreq.pem -out newpriv.pem          (ユーザの秘密鍵からパスワードを除去)
gw# cat newcert.pem newpriv.pem > newcertforqpop.pem     (これで qpopper 用の証明書ができた)
gw# cat > /usr/local/etc/qpopper.conf                    (qpopper 用の設定ファイル)
set clear-text-password    = ssl
set config-file            = /usr/local/etc/qpopper.config
set tls-support            = alternate-port
set tls-version            = default
set tls-server-cert-file   = /etc/ssl/newcertforqpop.pem
(Ctrl+D)
gw# echo 'pop3s stream tcp nowait/15/30 root /usr/local/pkgs/qpopper/sbin/popper \
    popper -s -f /usr/local/etc/qpopper.conf' >> /etc/inetd.conf  (inetd から呼ばれるようにする)
gw# telnet localhost 995                                 (ちゃんと動いているか確認)
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
^]
telnet> quit
Connection closed.

[NTP サーバの設定]
gw# ntpdate 133.100.9.2                                  (外部の NTP サーバと時刻の同期をとる)
gw# cat > /etc/ntp.conf                                  (NTP サーバの設定ファイル)
# clock.nc.fukuoka-u.ac.jp - 133.100.9.2
server 133.100.9.2
driftfile /etc/ntp.drift
(Ctrl+D)
gw# touch /etc/ntp.drift                                 (drift ファイルを作成)
gw# ntpd -p /var/run/xntpd.pid                           (NTP サーバの起動)
gw# echo 'xntpd_enable="YES"' >> /etc/rc.conf
gw# echo 'xntpd_program="ntpd"' >> /etc/rc.conf
gw# echo 'xntpd_flags="-p /var/run/xntpd.pid"' >> /etc/rc.conf
```

Mew は、POP over SSL をサポートしています。sslproxy というプログラムを使うのもよいかもしれません。Windows 用では wstone という SSL プロキシもあります。http://www.orangesoft.co.jp/wstone/

さて、いかがだったでしょうか。慣れていないうちは、一度にすべてのコマンドを打つのは苦痛そのものです。しかし、慣れてくるとたったの 2, 3 時間の作業で終ってしまいます。もちろんマシンパワーにもよりますが。

これを一通りできるようになれば、これだけの知識が得られるばかりでなく、快適環境が手に入るでしょう。後は、自分の好みにあわせて、特に HTTP サーバなどはチューニングが必要で、設定

ファイルの記述方法に関しても慣れておく必要があると思います。後、各種サーバのバージョンアップの際には、symlink の張りなおしを忘れないようにしてください。誤解のないように付け足しておくと、otsu-city.com は僕が個人的に所有しているドメインです。使うときは一報くださいね。

それでは、よいサーバ管理者ライフを。

2　簡単なHTTPクライアント

00230088 電子情報工学科 3 回生 春井 宏介

2.1　はじめに

コンピュータ間でのデータのやりとりにおいてその通信方法を定めたものをプロトコルという。ブラウザなどによるホームページ閲覧、インターネットメールのやりとりなどあらゆる通信についてそれぞれプロトコルが定められている。すなわち、ある特定の通信をするときには、それに対応するプロトコルに従う必要があり、プロトコルの存在しない通信はあり得ないのである。プロトコルは非常に多く存在するが、ここでは、ホームページを見るときに利用されている HTTP(HyperText Transfer Protocol) を、Perl による簡単な HTTP クライアントを作成しながら見ていこうと思う。

2.2　HTTP について

HTTP は、情報の転送を依頼するリクエストとそれに対するサーバのレスポンスから構成されている。URL、Web ブラウザの種類、使用言語などの情報を `GET` リクエストで Web サーバに送信すると、Web サーバはそれに応じてデータやエラーコードなどを返してくる。このデータのヘッダにはリクエストの可否、現在の時刻、サーバの種類などが書かれており、これと HTML などの要求されたデータ本体が返答されてくる。これにより Web ブラウザは、返信されてきた HTML ファイルを解析して表示する。

2.3　Telnet による接続

まず、Telnet を使って Web サーバにアクセスしてみる。Web サーバ（HTTP）は、通常、ポート番号 80 を使用するので、プロンプトで以下のようにホスト名とポート番号を指定することで該当する Web サーバにアクセスすることができる。

```
% telnet www.kit.ac.jp 80
```

これを実行すると、以下のように返事が返ってくる。

```
Trying 133.16.7.95...
Connected to wwwkit.kit.ac.jp.
```

```
Escape character is '^]'.
```

次に、見たいページのリクエスト (Request) をサーバに送る。

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

このように打った後、2 度 [Enter] を押す。2 度 [Enter] を押すのは、HTTP でリクエストがそこで終了であることを示すために、改行のみの行を用いているからである。すると、

```
HTTP/1.1 200 OK
Date: Mon, 7 Oct 2002 11:28:26 GMT
Server: Apache/1.3.26 (Unix)
Last-Modified: Wed, 26 Jun 2002 05:30:59 GMT
ETag: "5100a-a4-3d195193"
Accept-Ranges: bytes
Content-Length: 164
Connection: close
Content-Type: text/html

<html>
<head>
<meta http-equiv="refresh" content="0;url=./ja/index.html"></head>
<body>
<a href="./ja/index.html">京都工芸繊維大学ホームページへ</a>
</body>
</html>
Connection closed by foreign host.
```

と表示されるはずである。HTML の始まりの直前に空行があるが、このより前をヘッダ、後をボディといい、ヘッダ部分は HTTP での通信における各種情報、ボディ部分はホームページ本体（HTML）である。（このページには待ち時間 0 の refresh が設定されているのでブラウザから見ると、すぐに http://www.kit.ac.jp/ja/index.html へとばされることになる）

これがもっとも基本的な HTTP での通信である。

ここで送った “GET” のことをメソッド (Method) という。GET メソッドはファイルを送信してほしいという意味である。また、ファイルのリクエスト以外にも、HTML フォームからのデータも GET メソッドで送信することができる（GET メソッドの機能とは言えないが…）。例えば CGI に対して

```
/cgi-bin/sample.cgi?data1=kit&data2=kougei
```

という引数を渡したい場合は、

```
GET /cgi-bin/sample.cgi?data1=kit&data2=kougei HTTP/1.0
```

で実現できる。

HTTP/1.0 で規定されているメソッドには GET の他に HEAD、POST など数多くあるが、今回は取り上げないことにする。

HTTP クライアントでは Perl による簡単な HTTP クライアント作成してみることにする。ここでは Socket モジュールを利用するため、Perl5 を使うことにする。

```
1: #!/usr/bin/perl
2:
3: # Socket モジュールを使う
4: use Socket;
5:
6: # HTTP プロトコルを使う
7: $port = getservbyname('http', 'tcp');
8:
9: # ホスト名を IP アドレスの構造体に変換
10: $iaddr = inet_aton("www.kit.ac.jp");
11:
12: # ポート番号と IP アドレスを構造体に変換
13: $sock_addr = pack_sockaddr_in($port, $iaddr);
14:
15: # ソケットの生成
16: socket(SOCKET, PF_INET, SOCK_STREAM, 0);
17:
18: # $sock_addr に従って接続
19: connect(SOCKET, $sock_addr);
20:
21: # SOCKET のバッファリングを OFF にする
22: select(SOCKET); $|=1; select(STDOUT);
23:
24: # HTTP リクエストの送信
25: print SOCKET "GET /index.html HTTP/1.0\r\n";
26: print SOCKET "\r\n";
27:
28: # ヘッダ部分の確認
29: while (<SOCKET>){
31:  m/^\r\n$/ and last;
32: }
```

```
33:
34: # ボディ部分の表示
35: while (<SOCKET>){
36:  print $_;
37: }
```

コード説明

```
7: $port = getservbyname('http', 'tcp');
```

getservbyname はサービス名からポート番号を取得する関数である。これは、/etc/services など（Windows なら C:¥windows¥services、または %SystemRoot%¥drivers¥etc¥services）を参照して行われる。'http', 'tcp' で「TCP/IP の HTTP サービス」ということを示している。

```
10: $iaddr = inet_aton("www.kit.ac.jp");
```

inet_aton はホスト名を表す文字列を受け取り IP アドレス（4byte の構造体）に変換する関数である。IP アドレスは OS が DNS サーバに問い合わせて収得する。ちなみに、inet_ntoa 関数により、この$iaddr から 133.16.10.95 などのよく見る文字列に変換できる。

```
13: $sock_addr = pack_sockaddr_in($port, $iaddr);
```

pack_sockaddr_in は、IP アドレスとポート番号をまとめて一つの構造体にする関数である。

```
16: socket(SOCKET, PF_INET, SOCK_STREAM, 0);
```

socket はソケットを生成する関数である。UNIX では、ネットワーク経由のデータのやりとりには、必ずソケットというものを使う。PF_INET とはインターネット利用を、SOCK_STREAM は TCP 接続をそれぞれ表す。ソケットを使うとファイルの読み書きと同様の書式でネットワーク経由のデータのやりとりができる。すなわち、

```
socket(SOCKET, …)
```

でオープンし、

```
print SOCKET "…";
```

でデータの送信ができる。

```
19: connect(SOCKET, $sock_addr);
```

pack_sockaddr_in 関数で生成した$sock_addr に基づいて接続（connect）する。

```
22: select(SOCKET); $|=1; select(STDOUT);
```

ソケットに対するバッファリング機能を OFF にする。効率のよい入出力のため、UNIX にはバッファリング機能がある。入力された文字列をバッファに蓄積する機能だが、ソケットを流れるデータを OS がバッファリングすると通信がうまくできなくなる。

この時点で、前述の

```
telnet www.kit.ac.jp 80
```

の操作が終了したことになる。次に、HTTP での GET リクエストを行う。

```
25: print SOCKET "GET /index.html HTTP/1.0\r\n";
26: print SOCKET "\r\n";
```

ソケットに対してデータを送ります（＝ Web サーバへのデータ送信）。なお、HTTP において、プロトコル上の改行は \r\n（CRLF）と定められているので、すべて “\r\n” を用いている。

Web サーバへのリクエストが完了したので、受信したデータの受け取り作業に入る。

```
29: while (<SOCKET>){
31:  m/^\r\n$/ and last;
32: }
```

受信したデータのヘッダ部分の処理である。31 は改行のみの行ならループを抜けるという処理である。

```
35: while (<SOCKET>){
36:  print $_;
37: }
```

改行のみの行以降のボディ部分を表示する処理である。

なお、Telnet でのアクセスからも分かる通り、HTTP ではヘッダとボディの送信が終了したらコネクションを切断するため、切断・終了などの処理の必要はない。それゆえ、例えば受信した HTML 内に画像などの挿入があると、画像の数だけ connect する必要があるのである。ただし、HTTP/1.1 では接続を維持することもできるようになっている。

これを実行すると、ヘッダ部分が除かれた http://www.kit.ac.jp/index.html が表示される。

以上で、簡単な HTTP クライアントの作成・実効を試してみることができた。

2.4 実用的な HTTP クライアントへ

今回作成・実行してみた HTTP クライアントでは URL が固定されているなど実用にはまだまだ遠い状態である。GET 以外のメソッドへの対応、多くのヘッダ処理、エラーコードに対する処理などを追加しなければならない。そして、HTTP ファイル転送や HTML 変換 Proxy などの作成が今後の課題である。

3　P2Pによるファイル共有の楽しみ方

00230098 電子情報工学科 3 回生 松村 宗洋

3.1　なぜファイルを交換するのか？

という素朴な疑問にぶち当たるのは、なにも最近流行っているインターネットを介したファイル交換ソフトを語るときだけではない。インターネットという情報交換の場が提供されるのと同時に、ファイル交換の場というのも提供されてきた。日本においても外国においても、Warez を筆頭とするいわゆるアンダーグラウンド[1]というものが人間の物資欲とある種の**オーガズム**を掻き立ててきたという歴史があるということは自称パワーユーザという人ならご存知の方がいらっしゃるかもしれません。ソフトのコピー行為は違法である。周知の事実。しかしこのコピー行為を助長するためのさまざまなツールや環境が同時に開発され、提供されてきているのである。人間は決して性善説では語れない証明でもあると言い切っても過言ではないかもしれない。最近の読者の大半の方は P2P ソフトウェアによる個人ファイルの交換に慣れていることだと思いますが、ここで軽くファイル交換としていままでインターネットを利用してきたアンダーグラウンドな人類が取った方法を箇条書きしてみるとしましょう。

- 広帯域なサーバ (大学・研究機関など) の Anonymous FTP によるファイルのアップロード、ダウンロードによる配布
- 無料 Web スペースによるファイルのアップロード、ダウンロードによる配布
- ICQ や IRC によるファイル転送機能を利用した交換
- 無料 Web ストレージサービスを利用したファイルのアップロード、ダウンロードによる配布
- 自宅サーバの FTP による IP アドレス告知型の交換

さて、ここまではいいとしましょう。ところが彼らにとって、このファイル交換という違法性の高いやりとりには当然リスクが伴うわけである。**「逮捕」**である。上記に箇条書きした方法では、警●[2]関係者でも簡単に交渉又は DOM[3]として、ファイルを取得できてしまい、証拠物件として簡単

[1] underground：裏社会。麻薬、覚せい剤、銃器、という分野のほかにも、ポルノ、違法コピー、偽札造りと表社会では一般的に違法だとされる分野の総称
[2] 刑事的違反を調査、違反者の拘束、民事的な訴訟などの調査 を行う関係当局
[3] Download Only Members の略。ダウンロードだけをして御礼の書き込みなど何もしない人たちのことをいう。一般的にアップ職人には嫌われる。

に立件してしまうことができるのです。これではファイルを提供する側も枕を高くして寝れなかったでしょう。そこで、彼らの祖先は次のようは発想をしました。

「ファイルを偽装圧縮して初心者や警●が取得できないようにしよう」

そこでさまざまな圧縮技術が生まれました。日本のインターネット事情からもわかるように、ADSL のような高速通信が可能な方法が現在のように定着するまでは、遅いくせに金だけかかるし電話代までかかるというボッたくりなサービスの ISDN というものが推奨されていた時期がありました。この通信速度の遅さをカバーするためにもファイルサイズをできるだけ小さくするということも圧縮をかける上では重要なことでした。

くだらない名前が付けられた圧縮規格がたくさん生まれました。ただその中でも RAR や CAB[4]を超える高圧縮率の規格も生まれました。このようにして彼らの祖先はファイルを乱立した圧縮規格を多段に適用し、どのアーカイバを使って解凍すればいいかということに躍起になって努力し時間を費やしてきました。(英語圏の外国人はそのような発想をしなかったようだ。どんなファイルも分割や 1 つにまとめるといっただけの目的にしか圧縮は用いなかった) アップロードする人 (敬意を称して**アップ職人**と呼ばれる) やダウンロードする人の間には Give and Take の関係がなければならず、アップ職人もそれが確認できなければならないのでありました。提供側はもらった人に感謝の意を受け取り、ダウンロードする人はファイルを得ることにより利益を得ます。ここに**恩賞と奉公の関係に似た和の心**を垣間見ることもできましょう。ここまででもおわかりのように、人はファイルを交換することにより規模は別にして、**社会の一部でありたいという欲求を満たしていた**のです。ただしこのような閉鎖的な社会にも礼儀作法が存在します。この礼儀作法についての詳細は検索エンジンなどで探して心得るなり好きにしてください。

しかし限界がきた

警●関係の人におよばず、ACCS など**著作権保護のための団体**も パワーユーザーを調査時に雇うようになりこの圧縮による偽装も簡単に回避され、「見せしめ逮捕があったぞ」というニュースが飛び交っていたのでした。もっと手軽にリスク無くファイル交換ができないだろうかと、アンダーグラウンドな日本人に限らず世界中の人がその想いを共有する時期でもありました。

3.2 一世風靡した Napster の楽園

これは人類の進化の歴史と言ってもよいのだろうか。ここで Napster[5]というファイル共有ソフトが開発されました。これは Napster の開発陣営が用意したサーバにクライアントソフトを使って接

[4]Cabinet(キャビネット)：Windows が標準でもちいている圧縮規格
[5]Napster http://www.napster.com/ 。現在は T シャツのロゴの受付サイトになっているようだ。

続し、接続したクライアントのハードディスク内の共有ファイルをお互い検索したり閲覧したり、ダウンロードしたりするようにしようとしたものである。各サーバ、クライアントが「Nap プロトコル」に沿った通信を行い ファイル交換を P2P[6]の接続で行えるというものである。このプロトコルが末永く使われることになろうとは誰も知る由もなかった。Napster の Nap プロトコルを簡単に説明すると以下の図 3.1 のようなものである。

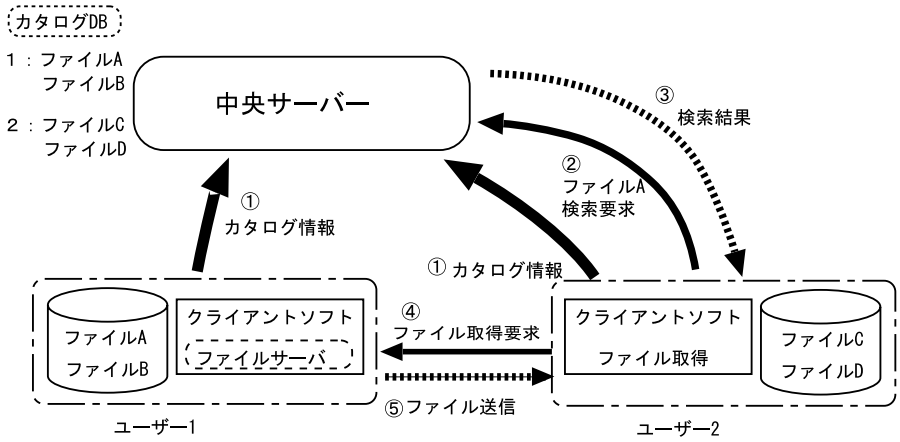


図 3.1: Nap プロトコルによる Peer-To-Peer ネットワーク網

1. Napster の中央サーバに接続したユーザそれぞれがカタログ情報、つまり各々のユーザのハードディスク内にどんなファイルがあるかの情報を送信して登録する。
2. ユーザ２がファイル A の所在についての検索を中央サーバのカタログ DB に対して行う
3. 中央サーバがファイル A を所持しているユーザマシンの情報をユーザ２に対して検索結果として返す
4. ユーザ２は直接ユーザ１にファイル取得の要求を出す
5. ユーザ２の要求を受け取ったユーザ１はファイルをユーザ２に送信する

このような手続きを踏み、誰でもクライアントソフトをインストールし、接続するだけで簡単に目的のファイルを交換し合えるという事態が起こってしまったのである。この状況はいままでアンダーグラウンドに手間を惜しんでファイルをシコシコと取得してきたウェアーズユーザにとっては涙がでるほどお手軽で嬉しい物であったでありましょう。

しかし楽園はつかの間だった

[6]Peer-To-Peer の略。広義では多数の関連ノードと互いに対等な関係でやりとりを行うということ。ネットワークに関して言うと、どのノードもサーバにもなりクライアントにもなり、お互いに上位下位の関係がなく対等に通信ができるシステムのことである。

それはどういうことかというと、Napster の欠点として図からもわかるように、 中央サーバに依存しすぎること が挙げられます。つまり中央サーバが止まってしまうと、要するに何もできないのである。はじめの認証を中央サーバで行う以上、何かの物理的障害、**外的 (権力的) 圧力**に非常に弱いのである。Napster はものの見事にその外的要因[7]に押しつぶされてしまったのでありました。

3.3 Never Never surrender ～Nap クローンの出現～

生物には存続するための能力があるのは誰もがご存知だと思いますが、一度作られた楽園生活を簡単に奪われて屈することはないものであります。言わなくても Nap クーロンが出現しました。クローンとは同じプロトコルや仕様を持つが、全く別の団体が本家本元のソフトウェアーを改造、改良してリリースしたソフトウェアーのことである。ファイル共有ソフトウェアクローンとして以下のようなものがあります。

- **OpenNap[8], SlavaNap[9]**

  Nap プロトコルを純粋に下位互換として持つ。OpenNap は UNIX 版、Java 版、Windows、MacOS 版とプラットホームが多彩。SlavaNap は Windows 上で詳細な GUI オプションでサーバの管理が容易に可能。チャットのチャンネルの機能も強化。

- **gnutella[10], Gnutmeg, FreeNet, iMesh**

  中央サーバを必要とせず、分散的に相互に接続したネットワークで情報交換し、ファイル交換は Peer-to-Peer で行う

- **DFSI**

  IRC 経由でファイル共有ができるらしぃ

- **Hotline**

  マックでのファイル交換のツールとしてかなり有名なソフト。Windows 用も出たが日本語対応が遅れるなどであまり流行らなかった。

- あともろもろ ···

  KaZaA、Fileswap、Jaunglemonkey Phosphor、Konspire(Java で動く)

たくさんクローンが生まれる中、私たち日本人の中で有名になったのは **Gnutella** や **OpenNap、SlavaNap** であろう。Gnutella は今まで中央サーバに依存していたファイル共有ソフトととって替

[7] 全米レコード協会 RIAA による訴訟
[8] OpenNap: http://opennap.sourceforge.net/
[9] SlavaNap: http://slavanap2.sourceforge.net/
[10] http://www.gnutella.com/

わって出現したものである。Gnutella の用いたプロトコルは非常に興味深いものがある。簡単に説明しましょう。(図 3.2)

Gnutella は沢山の**サーバント**と呼ばれるネットワークの接点 (Gnutella ユーザ) が相互に接続をしている通信網上で、Peer-to-Peer でお互いに接続をしていくというのが基本スタイルである。そのため全サーバントは完全に対等な立場にあり、接続元、接続先というものが静的には決定できない。

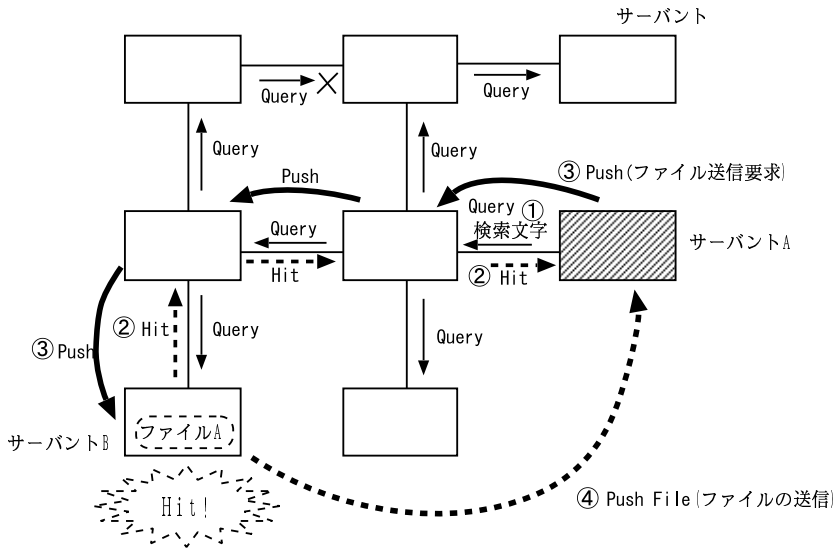


図 3.2: Gnutella の Peer-To-Peer ネットワーク

1. サーバント A が 1Hop 隣のノードに Query 識別子 (検索文字列) を送信する
2. Query を受け取った各サーバントは自分のディスク内に該当するものがないかを検索し、あれば Hit(検索結果) を送信元サーバントに返す。また、Query の TTL を減らしペイロード ID を付加し、送信元以外の 1Hop 隣のサーバントにフォーワードされる。
   サーバント B において Hit したとする。サーバント B は Hit 識別子を Query 識別子を送信し、各サーバントは Hit 識別子を Query 識別子がルーティングされた同じ経路に沿ってのみ送信する。
3. Hit 識別子を受け取ったサーバント A はファイルダウンロード要求である Push 識別子を今度は Hit 識別子が通ってきたものと同じ経路に沿って送信させます。
4. Push 識別子を受け取ったサーバント B は今度は直接サーバント A に対してファイルを送信します。ファイルのやりとりにはなんと HTTP プロトコルを使用します。

Gnutella のような Peer-To-Peer ネットワーク網はインターネットの姿と非常に似ているといえます。インターネットを使った接続方法といえば、普通大多数の人がサーバ・クライアント方式のことを頭に思い描きます。ですが、サーバが無ければクライアントは何もサービスを受けることがで

きません。またサーバの障害によってクライアントのインターネットリソースは失われてしまいます。そのような特定のサーバに依存しなければならない姿を脱却するべく広がったのが「インターネット」に他ならないのです。

単純なことですが、インターネットの基本的な仕組みを振り返ってみるとわかると思います。旧来大学や研究機関同士の専用ラインであったネットワークを中継する機器 ルータを一般 ISP へのネットワークに繋がる **NOC(Network Operating Center)** に多数はさむことによってどこからでもアクセスが可能なネットワークが広がっていったというわけです。このあたりの話はインターネットの歴史を振り返るとかならず語られることなのですが、その根本の姿を忘れインターネットにまで階級や権限による役割の差をはっきりさせたサーバ・クライアント型中心の思考を蔓延らせたことは、まさに時代を逆行してきたといえるでしょう。

しかし実際のところ、日本の回線事情が影響として大きかったのです。お世辞にも早いとは言えない ISDN 回線や高い接続料 (しかも時間による課金制) で常時接続とは程遠いものだったのです。P2P が成立するには常時接続が最低条件でもあります。そして回線の高速さも必須事項であります。日本は最近までこの条件がみたされていなかったのです。

3.4 WinMX と WPNP の登場

Napster や Gnutella もそれぞれデメリットがあった。Napster は中央サーバに検索負荷がかかる。Gnutella は各サーバントにルーティングのためののパケットや Ping,Pong の識別子が飛び交いネットワーク的な負荷が大きくなる。ここで **WPNP** の登場である。WinMX が登場した。WinMX が用いたネットワークのシステムは図 3.3 のようになっている (WPNP Ver2)。

1. ユーザが WinMX ソフトを使用して WPNP(WinMX Peer Network Protocol) 中央サーバに問い合わせに行く。中央サーバは自分にぶら下がっている親の接続ポイント情報をもっており、WinMX クライアントに渡す。受け取った WinMX クライアントはいずれかの親に子としてぶら下がるか、親として中央サーバに直接接続する。
2. 接続すれば WNPN サーバツリーの一部となり検索要求などを送信できるようになる。検索文字列を自分が子として接続しているなら、自分の親に渡す。親なら直接中央サーバに向けて検索文字列を流す。
3. 親は子からの検索文字列を中央サーバにルーティングし、中央サーバからの検索結果を子にルーティングする。
4. 検索結果を受けたクライアントはファイル所持者にファイル送信要求を出す。
5. ファイル送信要求を受信した WinMX ユーザは要求元にファイルを送信する。

この WinMX の WPNP のメリットがいくつかあります。まず、親と子との関係があること。この 2 段の木構造のため、ノードのトラフィックが多段階の木構造のネットワーク接続と比べて少ない。

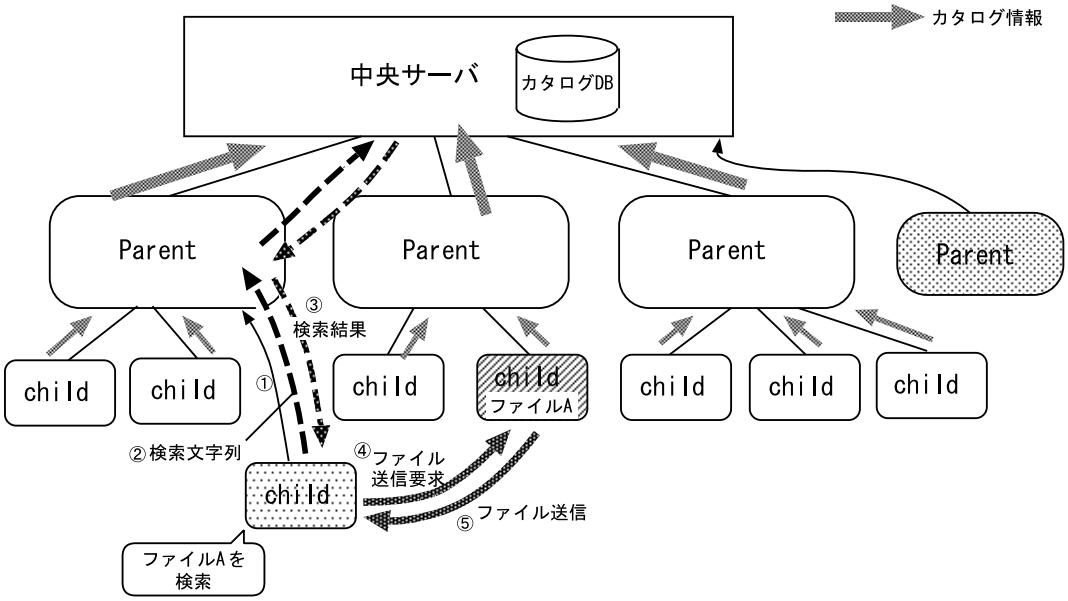


図 3.3: WPNP ver2 による Peer-To-Peer ネットワーク

そして接続条件を満たしているクライアントはいつでも親ノードとして WPNP に接続することができる。中央サーバから見た利点として、沢山の子からのトラフィックが中央サーバに一点に掛かることを防げる仕組みになっている。トラフィックを親ノードを通すことによって軽減しているのである。

また WinMX ソフトはいままでの Nap プロトコルにも対応している。OpenNap や SlavaNap で立てられたいわゆる**「子鯖」**にも WPNP と同時に接続し、検索をかけることができます。このお手軽さと Napster 譲りの検索力が人々を魅了し、広がっていたのである。(WinMX の存在を友達から聞き、パソコンを買ったという人もいるくらい)

ここで子鯖について少し説明をしておきましょう。日本における子鯖としては大部分が「２ちゃんねる[11]系」人達によってボランティアで立ち上げられています。サーバ情報やお知らせの告知は２ちゃんねるのダウソ掲示板で行われ、数人の有志を募ってサーバ同士をリンクさせ、リンクされたサーバ全体で平均 2000 人弱のユーザを収容させるくらいを目標として運営されている。サーバソフトウェアには OpenNap、SlavaNap が使われますが、元ソースは NullIP[12]化されていないため、IP アドレスがサーバコマンドの「whois」などで簡単に調べられてしまうために、２ちゃんねらーの手によって改良されたバージョンがいくつもリリースされました。2 ちゃんねる系子鯖への接続

[11] 2 ちゃんねる: http://www.2ch.net/ 巨大掲示板群サイト
[12] NullIP 化: 元ソースからコンパイルした OpenNap や SlavaNap は検索結果の相手の情報表示により IP アドレスがすぐに判ってしまう仕様であったが、whois のレスポンスパケットから IP アドレス部を 0.0.0.0 にしてしまうパッチを当てることにより IP アドレスが漏れないようにすること

の仕方を図で示すと図 3.4 のようになります。

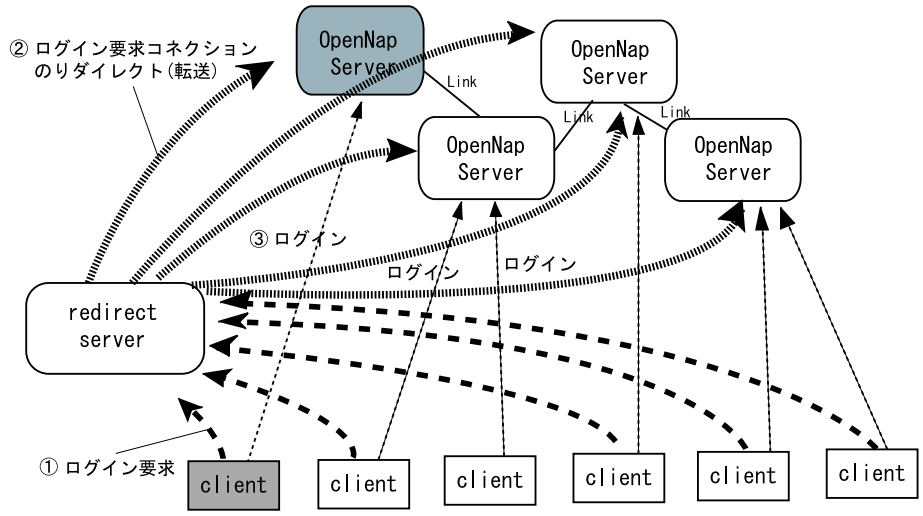


図 3.4: 2 ちゃんねる系 Nap 子鯖のネットワーク体制

まずクライアントは 2 ちゃんねらーの提供している OpenNap サーバの何れかに接続するのであるが、ここで提供者側は負荷分散のため、ランダムにリダイレクト (転送) させるサーバを用意しています。このサーバに通常のサーバへ接続する方法と同じようにログイン要求を送信すると、用意されたお互いがリンクしあって同期を取っているサーバのどれかに接続が転送され、通常通りログインすることができます。

このようにして WinMX ユーザは WPNP と子鯖を使い分けてファイル共有を堪能していました[13]。ところが、WPNP サーバ Versin2 が**閉鎖**となっています。すなわち、WinMX ver2.x のクライアントを使用して WPNP を使用することはできなくなってしまいました。これは WinMX 側が新しい仕様の WPNP ver3 のサーバを立ち上げ、今までのユーザにはその新サーバのほうに移行してもらおうということで、そうなったのでした。WPNP3 のサーバはまだ試験的な要素も多くユーザ数も Version 2 のころと比べると圧倒的に少なくなってしまいましたが、Versin2 と比べるとさまざまな改良がなされています。まず、いままでは多数の人から同じファイルをダウンロードするのにもすべて別のファイルとしてダウンロードする必要がありました。そしてレジュームも聞かないので大きなファイルをダウンロードしている最中の切断には泣かされるものがあります。そこで、

- 同一ファイルを分割して多数の人から同時にダウンロードできるようになった。
- レジュームが効くようになりました。
- 中央サーバに依存しなくなった。

[13] 2ch 系子鯖以外にも NapMX や Napigator などのツールを使用することにより、他の子鯖リストを WinMX に動的にインポートし、外国人の立てた子鯖を利用する方法も広く使われていた

3 項目目の依存性がなくなったのは非常に大きいことです。親サーバ同士がリンクを結びそれぞれの親が対等の関係になることにより中央サーバの必要性をなくすことに WPNP3 は成功しました。このあたりは Gnutella と似ていますが、WPNP で特徴的であった親子関係を維持しながら Peer-to-Peer ネットワークを構築しています (図 3.5)。

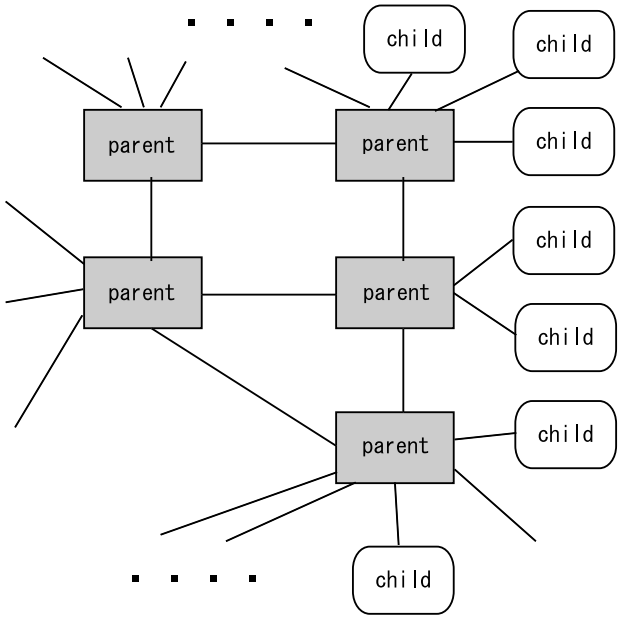


図 3.5: WPNP ver3 による Peer-To-Peer ネットワーク

ただ、Gnutella でも問題になりましたが、図 3.5 のようなネットワーク網では親にかかるネットワーク的負荷が多くなりがちで、なかなか検索結果がでなかったり表示が遅かったりとまだまだ改良の余地があります。今後に期待しましょう。

ざっと WPNP2、3 の仕組みは理解していただけたと思います。ですが、ファイル交換には大きな制約があります。それは**「著作権」**です。一般市民が何百メガ Byte もするその人の創作したファイルを普通持っていますか？ 交換するネタとして多いのが市販ソフトウェア、音楽 MP3 データや映画などの動画ファイルです。これらはほとんどが著作権で守られるべきものばかりだと思います。これらのものが交換されることにより市販の電子データが別人に無償で渡ること (つまり購入することなしにタダでソフトや音楽 MP3 が手に入ってしまう) がいとも簡単に実現されてしまいます。このような事情が、Peer-To-Peer なファイル共有が**アングラ (Underground)**[14]だと言われている所以なのであります。AOL が Gnutella を開発した Nullsoft 社をサイト開設からわずか 2 日のうちに買収して閉鎖させたという理由もわからないことはないですね[15]。いままでの交換は Peer-To-Peer

[14]「アンダーグラウンド」 の略
[15]現在は Nullsoft 社から独立した Gnullsoft 社によって Gnutella の開発が進められている

でファイル交換を行う際、交換先のノードの IP アドレス等がクライアント側でもわかってしまうということから、摘発するための証拠をそろえる側としても十分な土壌があるということを忘れないようにしなければなりません。WPNP も Version3 になってからは実際の交換が成立しないことには相手の IP アドレス等の情報を得るのは難しくなってきているのです。

そこで日本人が Gnutella の Peer-To-Peer により匿名性を高めようとしたもの作ろうとしています。**Winny**[16]です。Winny のモデルは Gnutella や FreeNet から取っていると作者は言っています。ただ、それぞれの利点を取り入れたものになっているといえます。Gnutella からは Ping-Pong や Query のパケットの流し方を採用し、FreeNet からは接続時の匿名性を採用しています。Winny ではファイル交換のときも他のノードを通過させ、そのデータをキャッシュとして残しておきます。これが Gnutella と決定的に違い一度参照されたりルーティングされたファイルは再び違うノードからリクエストされたとき、近くにある最短の Hops 数のノードから取得することが可能となります。また、ファイルの取得までをルーティングするということから、接続先、接続元がお互いわからないようになり、違法な交換をしていた場合の摘発がかなり難しくなってしまいます。一般的には、これからの P2P 技術はより**匿名**へということを重点として進められています

3.5 P2P の行く末

いかがでしたでしょうか。今までなんとなく使っていた P2P ファイル共有ソフトウェアによってこれほどまで社会現象が起こっているという理由もわかって頂けたのではないでしょうか。しかしこの P2P 技術をファイル共有の技術としてアングラの烙印を押したまま社会の隅に追いやるというのはいかがなのでしょうか。Peer-To-Peer のシステムは私たちの身近に最近になって多く導入されてきています。というか注目されていると言ったほうが正しいかもしれません。たとえば**「インターネットオークション」**は立派な Peer-To-Peer なシステムです。参加者全員がだれでも顧客にもなり供給者にでもなりえるのです。他に例を挙げると、エンロンという企業の取る需要と供給の流通形態も Peer-To-Peer なのです。今までの供給者から卸売り業者を通じて一方的に消費者の方に流れるといった需要と供給事情も変化しつつあります。このような動向が近年になって覗えるようになったのも、情報通信技術の発展が関係しています。インターネットの Peer-To-Peer の発展も高速回線なくしては実現しえなかったでしょう。その情報の即興性が現代社会の Peer-To-Peer の推進力になっているのは間違いないでしょう。P2P はこれまで続いてきたヒエラルキーを打破する革命的な技術なのです。保守派は当然全身全霊をもって阻止してくるでしょう。Napster はその 1 被害者にしかすぎません。彼らは著作権を侵害する環境構築を幇助したと訴えられたのです。そして今後ファイル交換を拠点として広げようとしていた情報交換のシステムを浸透させることを妨害されたのです。しかしアンダーグラウンドな人々を含め、私たちは P2P の世界を垣間見てしまったのです。この広がりは収束を知らないでしょう。これまでの歴史からも、事実、コンピュータ関係の最

[16] http://www.geocities.co.jp/SiliconValley/2949/

新技術はアングラから生み出されるものです。ゴルゴ 13[17]がどれだけ人のためになろうが感謝されようが、彼のしていることは殺人です。彼らの反社会的行動は取り締まられるべきでしょう。しかしゴルゴ 13 からは研ぎ澄まされた精神力と信念を学び取ることができるように、これからはそこで保守的にならず、自らの向上のために P2P 技術を利用するべきではないでしょうか。

まじめな話ばっかりでかたぐるしかったと思いますが最後にひとつ！

著作物を交換するのはやめましょう！

結局まじめな話です（藁

[17]著者 さいとう・たかお 作のコミック「ゴルゴ 13」の主人公の人物。デューク 東郷という仮名を持つが職業はプロのスナイパー。妥協を許さない信念と強靭な精神力に魅了される読者が多い。今年で連載 30 周年を迎える。

参考文献

[1] OpenNap protocol — http://opennap.sourceforge.net/napster.txt

[2] (社) コンピュータソフトウェア著作権協会 — http://www.accsjp.or.jp/

[3] WPNP — http://www.pcmedixwebs.com/download/WinMX_FAQ.pdf

[4] 実験：ネットワーク管理者のための Gnutella 入門 — http://www.atmarkit.co.jp/fwin2k/experiments/gnutella_for_admin/gnutella_for_admin_1.html

[5] 大谷卓史 亀井聡 高橋寛幸　共著 2001 COOL BIZ-PACE シリーズ 「P2P がビジネスを変える」翔泳社

4　ホームページ作成入門

02220077 機械システム工学科 1 回生 森本 勇次

4.1　初めに

皆さん、インターネットをしたことはありますか？恐らく大抵の人は経験があると思います。

では、ホームページを作ったことは？

こうなるとずっと減ってくるでしょう。確かに、見る側でも楽しいのですが、ホームページを作るのも案外楽しいものです。

ページのレイアウト、コンテンツの種類、背景、etc…全てが皆さんの思うがまま。世界で一つだけのホームページを、誰もが持つことが出来るのです。

ここまでを見て興味をもってくれた方がいれば幸いです。

4.2　まずはスペースの確保

そもそもホームページは、ページ自体はパソコンで作ることが可能です。しかし、そのページは外部には全く開かれていないので、他者が来てくれることはありません。では、どうすれば外の世界とつなぐことが出来るのか？

その答えは、「インターネットの中に、自分のホームページを置く場所を作る」ことです。

例えば、私が自分のホームページを置いている「Tripod」、ここは検索エンジン「lycos」の管理するホームページスペースなのですが実は、「lycos」で自分のアカウントを持ってさえいれば誰でも自分のホームページを作り、置くことが出来るという素晴らしいところです。

それに、タグ（後々説明するが、ホームページ作成には欠かせない命令語）を覚えていなくても簡単にホームページが作れるという機能や、掲示板、チャット、カウンターもつき、しかも 12MB のスペースを無料で借りることが出来ます。（勿論、アカウント取得も無料）

いわば、「Tripod」は、初心者に非常に優しいホストサイト（自分がスペースを借りているページ）というわけです。

よって、初めての方はここから始めた方が良いでしょう。

4.3 ページの作成

さてスペースを手に入れたら次はページの作成ですが、それに際して、まず知っておかなければならないことがあります。まずは、拡張子を表示させておくことです。方法は、「スタート」→「コントロールパネル」（または「マイコンピューター」→「コントロールパネル」）→上の部分の「ツール」→「フォルダオプション」→「表示」と選んでいって、出てきた箱の中の「拡張子は表示しない」の横のチェックマークをはずしてください。

その次に、「タグ」です。ここではホームページを作るためのほぼ最低限のタグの説明しかしませんが、もっと知りたい方は検索エンジンで「タグ辞典」と検索してみればいくらでも出てきますので、自分にあったものを選んでください。また、HTML についてもっと知りたければ、HTML のタグの使い方などを詳しく説明したページがあるので探してみてください。

では、タグの説明です。なお、タグというのはほとんどのものが<…> </…>の形で表され、その間に挟んだものにタグの命令を適用します。

- <HTML>〜〜〜</HTML>…この間に挟まれた文章を HTML 言語で書かれた文章である宣言するタグです。これはページに一番最初に入れる命令で、後の命令は全てこの二つのタグの間に入れていきます。

- <HEAD>〜〜〜</HEAD>…ヘッダを表すタグです。この中に入れるのはタイトルを表すタグや検索ロボットに検索内容を伝えるタグを入れます。なお、この部分は実際のページ上には表示されません。

- <TITLE>〜〜〜</TITLE>…そのページのタイトルを指定するタグです。これによって今表示しているページのタイトルを指定できます。

- <BODY>〜〜〜</BODY>…この間に挟まれたものを表示する文章であると宣言するタグです。この間に挟まれた文章は<〜〜〜>の形のものをのぞいて全て、タグで挟まれていればその命令に従って、挟まれていなければそのままの形でページに表示されます。

- <P>〜〜〜</P>…間に挟まれた文字（文章）が一つの段落であると宣言するタグです。このタグで挟まれた文字（文章）の前後で一回改行します。

- <FONT>〜〜〜</FONT>…間に挟んだ文字（文章）の大きさや色などを変えたりするタグです。

- <FONT SIZE="?">〜〜〜</FONT>…間に挟んだ文字や文章の大きさを＋？する（？には数字が入る）

- <FONT COLOR="?">〜〜〜</FONT>…間に挟んだ文字（文章）の色を？色に変える（？には色のコードが入る）

- <A HREF="（リンク先の URL）">〜〜〜</A>…間に挟んだ文章を打ち込んだ URL にリンクさせるタグです。その文章をクリックするとその URL に飛ぶようになります。

- <A HREF="mailto:（メールアドレス）">〜〜〜</A>…指定したメールアドレスにメールを送ります。
- <IMG SRC="（画像のファイル名＜拡張子含む＞）">…指定した画像を表示するタグです。また、SRC="〜〜〜"を HREF="（画像の URL）"に入れ替えることによって、指定したリンク先の画像を表示させることも出来ます。
-
…改行を示すタグです。HTML の文章ではただ単に改行しても実際の文章では改行されないので、このタグを使って改行します。

ページが完成したら、保存してフォルダの拡張子（ファイル名.txt の下線部分）を html に変えてください。注意文が出てきますが気にせず OK をクリックします。アイコンが変わったら成功です。

4.4 自分に合った FTP クライアントを取ってくる

次は（必要な人は）自分に合った FTP クライアントを取ってきましょう。インターネットエクスプローラー 5 やネットスケープ 4．5 などには最初から FTP クライアントが入っていますので、そちらを使うなら新しく取ってくる必要はありません。

なお、FTP とは、インターネット上からファイルを取ってきたり、また、インターネット上にファイルを送ったりする際の手順を規定するもので、実際にそれを実行するものが FTP クライアントであり、これには非常に沢山の種類があります。

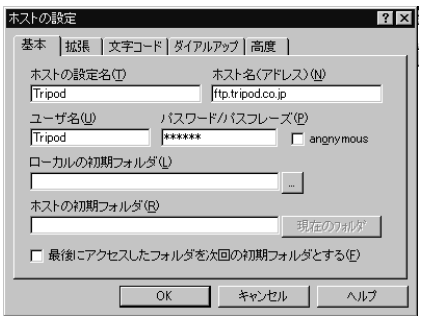
私がお勧めするのは「FFFTP」というフリーソフト（無料配布のソフト）の FTP クライアントです。これは、初心者にも分かりやすい簡潔な形をしているので、初心者の人にも非常に使いやすいと思います。私は初心者ですが、非常に快適に使わせてもらっているので、お勧めです。以下は、全て FFFTP についてのものです。

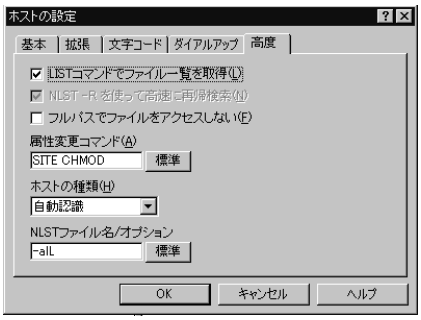
4.5 FFFTP の設定

FFFTP をダウンロードしたら、次は設定です。
下の手順に従って設定をしていってください。

1. FFFTP を起動して、「接続」メニューから「ホスト設定」をクリックしてください
2. 画面右上部に表示される「新規ホスト」ボタンをクリックしてください
3. 「基本」画面が表示されるので、下記のように設定してください（ユーザー名とパスワードはあなたの Tripod のメンバー名、パスワードに変えてください）

   ホストの設定名　Tripod
   ホスト名（アドレス）　ftp.tripod.co.jp
   ユーザー名　あなたの Tripod のメンバー名（例：Tripod）
   パスワード　あなたの Tripod のパスワード





ホストの初期フォルダ　（設定しないでください）

4. 次に「高度」タブをクリックしてください。「LIST コマンドでファイル一覧を取得」にチェックをつけ、「OK」ボタンをクリックしてください。

5. 新しく「Tripod」というホストが作成されていれば設定は完了です。接続ボタンをクリックして、アップロードを試してみてください。

ここで、ファイル名などは全て半角文字でないといけないので注意してください。ひらがななどの全角文字が入っているとアップロードできません。

これで、あなたのスペースにあなたの作ったホームページが送られました。こうして、外部から人がやってこれるようになったわけです。

4.6 最後に

こうして、あなたのホームページは一般に公開されました。人が集まるも集まらないもあなたのページ次第です。認められればあなたのページはどんどん見に来てもらえるので、それを目指して頑張ってください。

かく言う私もいまだに発展途上です。まだまだ勉強することも沢山あります。大変ですが、その分やりがいがあるので、日々勉強勉強です。皆さんも、興味があればホームページを作ってみましょう！

第III部

ソフトウェア

1　工芸的プログラミング ～しょにょサンー～

00230042 電子情報工学科 3 回生 越本 浩央

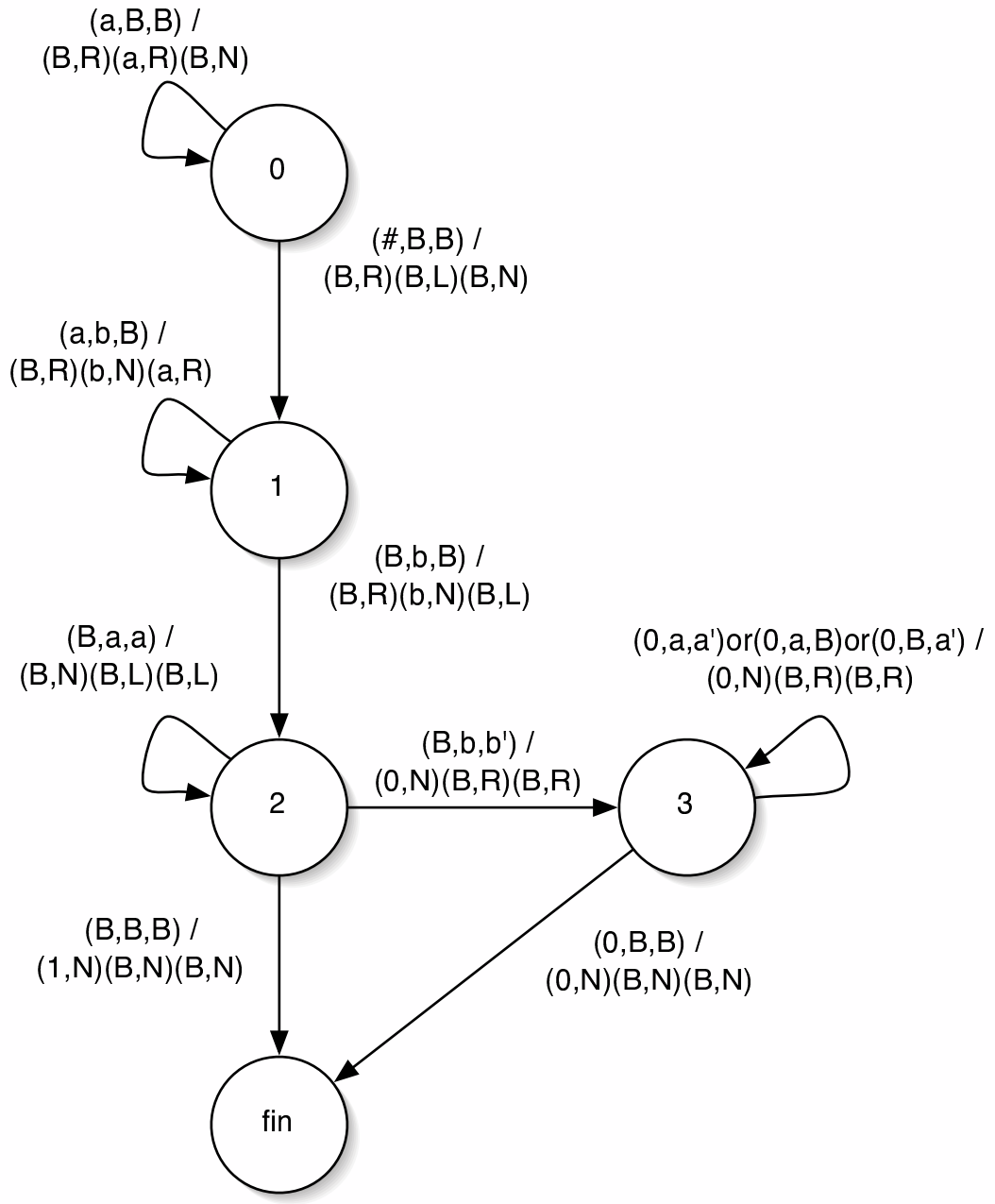
1.1　終わりに

　わかってるよ、こんな旧式のテキスト送るなんて田舎臭いってことくらいさ。俺としてはとにかく、あんたがこのメッセージを読まずに消しちまいませんようにって祈るっきゃないね。

　さて、どこから話したもんかな。こいつは今となってはどうでもいいようなことなんだけど、例の仕事 [ビズ] を持ち込んで来たのは俺じゃなかったのさ。軍用の危機管理メタビルダとチバシティにも出回ってねぇ防御プログラムなんて、ジョニィが最も好みそうな薬 [ドラグ] だとは思ったけど、あまりに出来過ぎた話だ。それに依頼主が財閥 [ザイバツ] だって聞いたら尚更のことさ。しかしまぁ、覆水盆に返らず [It is no use crying over spilt milk. by BORLAND]。あれはそう、虫が大量自殺を企てていた暑い夜のことだ。

1.2　停止問題の解決不可能性の鑑賞会

　「見てみろよ、ラルフ」
そういってジョニィは操作板 [デッキ] の上で繊細 [テクニカル] に指を走らせ、オートマトンを空間投影 [ソリプロジェクション] してみせた。



「自己同一性でも表明したいのか? でなければ、DNA 配列でも調べるさ」
「違うよ。僕が言いたいのはビジネスの話さ」
予定通りだ、と言わんばかりのうすら笑いを浮かべてラルフの方へ向き直り、黒々とした珈琲 [スピュー] を一口すすった。
「この原始的な計算譜 [プログラム] でさえ、スタックプッシュとスタックポップが潜んでる。そして逆にどんな複雑なコードでも結局はオートマトンと変わらない。所詮はチューリング機械さ」

回りくどい口調に些か苛立ちを覚えながらも、次に言うことの想像がついたラルフは、しかし黙ってジョニィを見つめた。
「それに比べてどうだ、こいつの芸術的な美しさは。でも当のチューリング自身がとっくに知っていたことだけどな」

```
((lambda (x) ((lambda (haltsq) ((lambda (z) (if (funcall haltsq
 z) (eval z) nil)) (print '(,x ',x)))) '(lambda (sexp) nil)))
 '(lambda (x) ((lambda (haltsq) ((lambda (z) (if (funcall haltsq z)
 (eval z) nil)) (print '(,x ',x)))) '(lambda (sexp) nil))))
```

「僕らもそれなりの芸術ってやつをやる頃だよ。だって考えても見ろよ、全身を駆け巡る直感が現実基底 [リアルスペース] の限界を示している、なのに先月からずっとピッツァしか拝めてないんだよ!」
怒気を含んだ最後の音節を言い放ったジョニィに、ラルフがやれやれといった風に述べる、
「その仕事 [ビズ] はおすすめ出来ない。しかしお前が実行することは分かってる。だから言っておくんだが、報酬の出所は Java.The.Hutt だ。ロクな仕事 [ビズ] じゃねぇぜ。気を抜くなよ。俺の命の保証をしろってことだがな」
当然といった顔でジョニィはコンソールに顔を戻していった。

1.3 コンパイル時に解決される二三の事柄

ミスは許されない。どんな仕事でも無視出来る誤差はある。だが今回に限ってそれは極限的に小さい。ラルフは決行前夜が近付くに連れてその思いを強めていった。

ジョニィは危険すぎる。過去の経験から言えば、今回も kewl なトリックを思い付くに違いない。だから今回はいけなかった。ジョニィが自分宛の量子 [Q-メカニカル] メールに目を通した時点で、既に最悪の選択がなされていた、これ以上は転び用が無い。そう自分に言い聞かせながらラルフは考えた。

二時間後、つまり決行十六時間前、ラルフの紡いだコードはこんなものだった。

```
#define STATIC_CHECK(exp) { char unnamed[(exp) ? 1 : 0]; }

template<class LongShot, class ByWhom>
LongShot safeGamble(ByWhom whom)
{
    STATIC_CHECK(sizeof(ByWhom) <= sizeof(LongShot));
    return reinterpret_cast<LongShot>(whom);
}
```

もしも危険を感知すれば、このコードはジョニィが行動を表明した時点で働き、エラーを返す。だがラルフはもう一つのコードを書いておいた。これがたった一つの冴えたやり方だということが分かっていたから …

```
struct EmptyType { };
class NullType { };

template<class T, class U>
struct ConsType {
    typedef T Car;
    typedef U Cdr;
};

template<class E, int N>
class Chain : public Chain<E, N - 1> { };

template<class E>
class Chain<E, 0> : public NullType { };

template<class Cons, template<class, int> class Unit>
class Fate;

template<class E1, class E2, template<class, int> class Unit>
class Fate<ConsType<E1, E2>, Unit>
    : public Fate<E1, Unit>
    , public Fate<E2, Unit> {
public:
    typedef ConsType<E1, E2> Cons;
    typedef Fate<E1, Unit> LeftBranch;
    typedef Fate<E2, Unit> RightBranch;
};

template<class AtomicEvent, template<class, int> class Unit>
class Fate : public Unit<AtomicEvent, NOW> {
public:
    typedef Unit<AtomicEvent, NOW> LeftBranch;
};

template<template<class, int> class Unit>
class Fate<NullType, Unit> {
};

typedef Fate<
```

```
    ConsType<ConsideredEvent1, ConsType<ConsideredEvent2
    , ConsType<ConsideredEvent2, NullType> > >
    , Chain> BizFate;
```

1.4 ZEN モンドー

ジョニィがデータヘブンの認証エージェントに接続をはかろうとしていた時、俺はジャンク屋のボルタックの親爺から購入したユーザキーでデータヘブンに接続し、ある要請 [クエリ] を発行しようとしていた。

状況開始の三時間前に取り交わされた段取りはこうだった。俺が一足先に正規ルートで (データヘブン) 内部に入り、構造体 [ストラクチャ] にある建造物の建築を要請する。この要請 [クエリ] は、次のようなオブジェクトジェネレータを作るものだった。

1. 認証エージェントのインスタンスをジェネレートすると、ジョニィをスーパーユーザとして認証するエージェントを生成する
2. オブジェクトジェネレータのインスタンスをジェネレートすると、そのオブジェクトコードに 1 と 2 の性質を追加してジェネレートする

こいつをオブジェクトスプーフィングを喰らわせた上で、走らせる。メモリオブジェクトの自己診断エージェントが次のサイクルで走ったときには跡形もなく消えてしまうが、その直前にジョニィはスーパーユーザとしてデータヘブンに実体化される手はずだ。

蒸し暑い高層マンションの一室で、二人はナノセカンド単位で状況を進めていった。
現実基底 [リアルスペース] の右手には、
聖痕 [スティグマ] よろしく電極 [トロード] が反応速度無限大。
頭部に接続されたデジタル単眼 [モノアイ] がネットスフィアのイメージを高速脳内転送。
量子力学 [Q-メカニカル]ICE ブレーカが作戦のシーケンスグラフに合わせ、
戦略・戦術をネットスフィアに実体化。
オブジェクト・アクティベート。
「ここは上級管理者の権利を持っていなければ接続出来ません」
[Unless the point haves a an upper administrator’s right, then it’s not connectable]
1 ナノセカンドの狂いもなく二人はデータヘブン中枢のターミナルに到達していた。

データヘブン内の全ての構造にアクセス出来る要請 [クエリ] が、ここでは出来る。ジョニィはインターフェースに次のような用件を告げた。「データヘブンの現在の全情報量は?」
「およそ 1374ExaBytes です」
「それじゃあ次の情報を検索してくれ。1374ExaBytes 以下のコードが終了する確率 Ω の値」
「要請 [クエリ] を受け付けました。ただいまから計算します。優先度を設定して下さい」
「下位エージェントと同じレベル」
「Accept!!」
インターフェースは消え、検索中を示すアイコンが中に浮かんでいる。これで仕事 [ビズ] は終わり

だ。あとは運を天に任すのみ。ジョニィは俺の方を振り向き、そのアニメのキャラクターを模倣したグラフィックスをにやっと笑わせた。

1.5 初めに

センタ内の中枢に位置する管理システムの CPU リソースメータの群れが、ことごとく振り切っている。そのほとんどが同じプロセスに属するスレッドを処理している。近くにおいてあるプロセスモニタウィンドウが、そのモニタの出力に切り替わった。

バイトコードが高速にスクロールしていく。同時に View ウィンドウが内容を翻訳していく。「断っておかないといけないことがある。俺とジョニィはある依頼を受け、そして実行し、成功した。こいつはあの要請 [クエリ] が出力している一つの検索結果だろう。巨大な量で音声や映像、プログラムなどが紡ぎ出され、また少し違った派生を生んでいる。サルのシェークスピアも悪くはなさそうだ。そのほとんどは大量のノイズにやられてゴミのようなものに仕上がっているようだが、今もデータヘブンのあらゆるリソースが他の可能性を模索しているところだ …

2　なぜ JavaScript なのか？

00230036 電子情報工学科 3 回生 岸田 匡司

はじめのはじめに

これは、普段それほどコンピュータと縁のない人を対象として書きました。なので、それ以外の人が読んでも面白くないと思います。逆に該当する方は、最後まで読んでいただければ、ちょっとした豆知識が得られるのではないでしょうか。

2.1　はじめに

タイトルを読んだ方は以後の文章にどのような期待をされるでしょうか？普通ならば「なぜ JavaScript が使われるのか」といったことや、そこから派生して「JavaScript の詳細」について書いてあるのではないかと思われるかもしれません。しかし、以後の文章はそんな格好のいいもの（？）ではありません。そんなことを書けるほど、今の私には知識もなければ時間もないのです。ここで私が書きたいことはただ一つ。Java と JavaScript の関係です。これらについてよく知らない人が聞いたなら、「JavaScript=Java の一種みたいなもの」と思われるのではないでしょうか？実際、私も初めは同じような認識でしかありませんでした。しかし、それは違うのです。Java と JavaScript は名前はたいへん似ているが非なるものなのです。この点について私の知る限りを書きたいと思います。ところで、ここで一つの疑問が湧き上がります。なぜ、JavaScript なんて紛らわしい名前を付けたんだ？

2.2　なぜ JavaScript なのか？

前節で Java と JavaScript は似て非なるものであると書きました。しかし、ここまで似た名前を持っているにも関わらず、この２つがまったくの無関係であるとは考えにくいでしょう。実際、この２つはまったくの無関係というわけではないのです。　さて Java と JavaScript の話をする前に、皆さんは「LiveScript」というものをご存知でしょうか。LiveScript は Netscape Communications 社によって開発されたプログラミング言語で、この LiveScript が JavaScript の前身になったものなのです。つまり LiveScript から JavaScript へと名称が変更されたのです。ここで問題としたいことは、名称変更そのものではなく、なぜ JavaScript という名称に変更されたのか、ということです。

Java を開発したのは Sun Microsysytems 社です。一方 LiveScript を開発したのは、前述したよう

に Netscape Communications 社です。このことからまず、開発した会社そのものから違うということが分かります。この点だけから考えれば、なおさら先の問題が不思議に思えてきます。この問題解決の鍵を握っているのは、Sun Microsysytems 社と Netscape Communications 社の技術提携です。LiveScript は Netscape 2.0 に実装されたのですが、時を同じくして Java が発表され、そしてそれほど時間を空けずに 2 社は技術提携を行ないました。このときに Java がたいへん話題になっていたため、その人気にあやかろうとマーケティング的な理由により、LiveScript から JavaScript へと名称が変更されたのです。実際には、LiveScript を元に Netscape Communications 社と Sun Microsysytems 社との共同開発によって JavaScript は開発され、またそのときに Java の構文を一部採用しました。

2.3 何がどう違うのか

JavaScript の名前の由来のところで、Java と JavaScript は開発元が違うと書きました。このことから初めに挙げた「JavaScript=Java の一種」ではないということは明らかであると言えます。そこで最後に２つの具体的な違いを挙げてみたいと思います。一番分かりやすい違いは、JavaScript はその名の通りスクリプト言語であるという点です。スクリプト言語であるので、インタプリタと呼ばれる言語プロセッサ（もとのプログラムを機械の分かる言語に翻訳するソフトウェア）により１命令ずつ解釈しながら実行されます。一方 Java はコンパイラと呼ばれる言語プロセッサにより全命令を一度に解釈して、その後 Java 仮想マシン（これについてはここでは触れませんが、これこそが Java の大きな特徴と言えるでしょう）によって実行されます。他にも違いはありますが、この点が最も大きな違いと言えるでしょう。

2.4 おわりに

時間がないこともあって、かなり勢いだけで書いてしまいました。至らない点が多々あるかと思いますが、ご容赦願います。これを読んで、「へぇー、そうなんだ」くらいの感想を持っていただけたら幸いです。

3　コンピュータ言語概論

00230111 電子情報工学科 3 回生 山本 大介

3.1　言語

3.1.1　概要

コンピュータの使い方を規定できるのがプログラムだけど、プログラムの書き方、つまりコンピュータ言語はたくさんある。有名なところだと、BASIC, C と言ったところだろうか? また、コンピュータ言語には文書構成を記述する言語もあり、HTML, TEX などがある。ウェブデザインとかでよく使われる JavaScript や Flash もほとんど一般のコンピュータ言語と変わらないほどよくできている。いずれの言語もある文字列や記号に特別な意味をもたせてコンピュータで処理させるようにし、変数などを使いその値を変更することによって汎用性を広げられるようになっている。

3.1.2　様々な言語

コンピュータ言語にもいろいろある。けど、基本的に英語から分化したもので、英語じゃない言語はないと言っていい。かつて JustSystem が日本語で指令するマクロを作ったりしたけど、そんな例はかなり希。JIS 規格の BASIC 言語も日本語だらけだったりするけど、これを実装してるの聞いたことない...。

3.1.3　言語設計の哲学

プログラマというのは不精で短気で傲慢じゃないといけないと言われてるらしい。([1]Programing Perl 参照)

- まず、プログラムは他人が読んですぐ解るように書かないといけない。みんなが難なく読めるような（可読性の高い）プログラムを書けたらそのプログラマは不精者になれるのだ。
- 次に、プログラムは人を待たせてはいけない。次にすべき作業を予測し、速やかな反応ができる（応用性の高い）プログラムが書ければ短気でも問題ない。
- そして、プログラムは他人にケチを言われないよう完璧にしないといけない。いちゃもんのつけようがない（保守性の高い）プログラムを書いてしまえば、誰にも文句を言われず傲慢

表 3.1: 言語と主な用途

機械語直訳	Assembly
汎用言語	BASIC Fortran (FORmula TRANslation system) COBOL (COmmon Business Orieuted Language)
汎用（関数あり）言語	C Pascal PL / I (Programming Language / One)
汎用（オブジェクトあり）言語	Python Java Ruby C++ C#
スクリプト言語	bsh (Bourne SHell) csh (C SHell) bash (Bourne Again SHell) awk Perl (The Practical Extraction Report Language) PHP
論理言語	Prolog (PROgramming LOGic) Common Lisp Scheme
ウェブコンテンツ言語	HTML (Hyper Text Markup Language) XML StyleSheet JavaScript Action Script
データベース言語	SQL

でいられる。

短い文章ではあるがコンピュータ言語の設計方針を大雑把に挙げると以上の 3 点にしぼることができる。どれを優先するかによってコンピュータ言語は分化を繰り返したと言えるかもしれない。

3.2 変数

最初にプログラムに汎用性を与えてくれた “変数”。その興味深い実装について書いてみることにする。

3.2.1 型

プログラムで必ず用いられる変数には、「型」という種類分けがある。言語によって、よりコンピュータのハード構造に近い「型」の種類分けであったり、意図的に型を隠している言語もある。

代表的なコンピュータ言語である C を例にとると「`int`(整数), `float`(浮動小数点型), `char`(1Byte 文字型), `short`, `long`, `double`」等とそれぞれの型の `unsigned`(符号なし) 型, ポインタ型がある。これはコンピュータのハード構造と密接に結び付いた型であり、多くの言語で似たような「型」をもっている。

ここで、独特な型を持つ言語をあげてみよう。まずは Java の `char` 型。これは C の `char` 型とは異なり 2Byte だったりする。これは Java の `char` 型が Unicode であり、アルファベットだけでなくひらがな、日本の漢字、ハングル、簡体字、繁体字などいろんな言語を同時に処理できるようになっているためだ。C のような 1Byte の変数は「`byte` 型」という別の型があり、C プログラマにとっては少しまぎらわしい。

次に Fortran。これにはなんと「複素数型」なるものがある。内部は浮動小数点型の変数がが実部と虚部で 2 つ分あるだけだが、平方根、べき乗等の計算をきちんとやってくれるので、数学のシミュレーション等にはかなり便利だ。

また、Java, JavaScript, Pascal, Fortran 等の言語には論理型 (`boolean`, `logical`) という真偽判定のためだけの変数[1]がある。言語のなかには条件分岐で論理型しか受け付けない言語もありこのような変数が作られたのだろう。

3.2.2 表記法

どの言語も英文字からはじまる英字と数字の文字列ならどんな単語でも変数にできるが、言語によって挙動が微妙にちがう。Fortran では `i,j,k,l,m,n` ではじまる変数は最初から整数型、それ以外は浮動小数点型となってたりする。型宣言をせずに最初から型が決まってるので便利だけど、例えば large という変数は整数型、small という変数は浮動小数点型となり、思わぬところで失敗したりする。

[1] 要するに `false` (0) と `true` (-1 or 1) にしかならない変数

BASIC や Perl, シェルスクリプト[2]では変数の型を変数を見ただけで判別できるように、記号をつけるようになっている。BASIC なら変数の後ろ[3]に、Perl なら前[4]に記号つける。この方式は変数を見分けやすくできる反面プログラムソースの見栄えが悪くなるという欠点がある。これはプログラマによって考え方もいろいろあるところかもしれない。

3.2.3 スコープ

変数で重要なのがスコープだ。プログラムが大きくなれば当然変数名がかぶってしまったりすることがある。それを避けるために長い変数名を使ったり、変数をリストアップしたりという努力するよりも、最初から特定の場所でしか変数が使えないようにしたほうが便利だ[5]。

スコープは変数が参照できる範囲のことで、グローバル (大域) とローカル (局所) がある。通常、ローカル変数は関数で外部から見られたくない変数に利用され関数などの特定のブロック内でし参照できない。グローバル変数はプログラム全体で共通して利用され、プログラムのどこからでも参照できる。

ただ、この分類にはあてはまらない微妙なスコープが Perl に存在する。Perl の `local` 関数だ。Perl の `local` 関数は local と言いながら、グローバル変数の値を保存する関数だ[6]。`local` 関数で指定されたグローバル変数は、指定されたブロック内でいくら変更されても、そのブロックから外にでると変数の値が元に戻る。挙動がローカル変数と微妙に似ているため、Perl プログラマの多くがこれが “ローカル変数” と思っている...。一般的なローカル変数は、同じ名前のグローバル変数とは全く関係がないので Perl プログラマの方はご注意を!

3.3 演算子、関数

ここで、どの言語にもある演算子と関数についてちょっと考えてみる。

3.3.1 演算子

いろいろな演算子

演算子は英語で言うところの operator なんだけど、operator を辞書で調べると、「操作する人、通信士、(電話) 交換手、(外科) 執刀医、経営者」とか書いてたりする。要するに、演算に限らずなにか操作をするものを指すわけだ。実際、コンピュータ言語の演算子は一般的に計算に限らず、さまざまな操作を含んでいる。

四則演算子の「`+,-,*,/`」、論理演算子の「`&,|`」、(言語によって違う) なんかは数学でもでてくるので演算といわれても違和感がないが、比較演算子の「`==,!=,<,>`」、ポインタ演算子の「`&`」

[2] bsh 等や awk 等のコマンド
[3] `%`は整数、`!`は単精度実数、`#`は倍精度実数、`$`は文字
[4] `$`は文字、`@`は配列、`%`は連想配列
[5] これは、変数だけでなく関数にも言える。
[6] Perl でローカル変数を指定するには `my` 関数を使う。

に至っては、「演算」と言うには違和感がある。どっちかというと「操作」のほうが合っていると言えるだろう。

言語別演算子

一般的に四則演算はどの言語でも「+,-,*,/」だが、代入演算子やべき乗、比較演算子は各言語微妙に異なる。代入演算子は通常「=」が用いられ以下のように書かれている。

```
a = b
```

この場合、b の値を a の値とするという演算であって、a と b が等しいという意味ではない。[7]「その意味をはっきりさせよう」ということで Pascal では

```
a := b
```

という数学で定義を表す演算子を用いる。また Lisp([2] 参照) では

```
(setq a b)
```

という変数と値の関係を意識した書き方になっている。条件分岐などで使われる比較演算子は言語ごとにバラバラで表 3.2 のようになっている。BASIC は簡易な言語のため代入演算子と等価演算子が同じ文字であったり、Fortran は記号でなく文字であったり[8]、言語ごとに特徴がみられる。しかし、直感的で利便性も高い C の演算子が Java, JavaScript, Perl[9]等、多くの有力な言語で利用されており事実上標準の演算子系になっているといえるかもしれない。

表 3.2: さまざまな演算子

演算子	BASIC	Fortran	C	Pascal
代入	=	=	=	:=
等価	=	.EQ.	==	=
大なり	>	.GT.	>	>
以上	>=,=>	.GE.	>=	>=
小なり	<	.LT.	<	<
以下	<=,=<	.LT.	<=	<=
不等	<>	.NE.	!=	<>

あと、余談となるがべき乗の演算方法は言語ごとにかなり異なる。C, Java は pow 関数でべき乗を計算するが、Perl, Pascal, Fortran, PL / I 等は「**」で, BASIC 等は「^」で, 大きく三通りがある。UNIX の gnuplot コマンドは「**」で、bc コマンドは「^」と UNIX 文化でも内部分裂していて個人的には「^」への統一を願っている...。

[7]かつては逆に a の値を b の値とするというような使われ方もあったらしい...。
[8]区切りにカンマを使っているため小数点と見分けづらい。
[9]Perl は通常は C と同じ比較演算子で、文字列処理の時のみ Fortran の比較演算子のコンマをとったものを利用したりしている。

演算子書法

演算子の書き方にはいろいろあるが、まず、一項演算、二項演算、三項演算、多項演算に分けられる。それぞれ、演算対象となる値の数で演算結果（返り値）は基本的にひとつだ[10]。

一項演算子はポインタ演算子（間接演算子、リファレンス演算子）や否定演算子等で通常前置演算子だ。例えばポインタ演算子は、

```
&a (C)   \$a (Perl)   ^a (Pascal)
```

という感じになる。二項演算子はほとんどの言語で以下のように中置演算子を採用している。

```
a + b    a = b    a && b
```

一部の言語はポーランド記法[11]、逆ポーランド記法[12]が採用されているが直感的でないのであまり使われていない。三項演算子は

条件式 ? 真の時 : 偽の時

という条件分岐の演算子以外ではあまり見られない。たぶん、これ以外にはないのだろう。

3.3.2 関数

一部の言語では関数は演算子の一部ということになっている。関数は前置多項演算子の一部というわけでだ。関数 (function) はサブルーチン (subroutine) やメソッド (method) とも呼ばれ、同じような作業をするプログラムを効率的に利用し、作業単位で区切ることによってプログラムを見やすくした。

多くの言語の関数は

```
function(var1, var2, ...)
```

という感じで、関数名の後に処理してほしい変数を引数として記述する。Scheme だと

```
(function var1 var2 ...)
```

となったり、BASIC だといろいろな文字で変数と変数を区切ったりする。

関数は 1 行で記述されるが、その 1 行で隠された多くの処理をすることになる。そのため、人間が直感的に理解できるよう名前等の見ため工夫したりすることが重要だ。このあたりになってくると自然言語の話ともからんでくるが、自然言語に近づけたコンピュータ言語はあまり成功していない。やはり、C のように記号を多用して論理的な記述を前面に押し出した言語のほうが書きやすくて、見やすいようだ。

---

[10] Perl の一部演算子は返り値が複数になることもある。
[11] `+ a b` のような記法。Lisp, Scheme で使われる。
[12] `a b +`のような記法。FORTH で使われる。

3.4 構文

ここでは、言語の全体像や流れを規定する構文（主に手続き型言語）について考えてみる。

3.4.1 条件分岐

プログラムに条件分岐のないものはない。変数の値の大小で処理内容を変えていくのがプログラムの目的だからだ。現実の事象を数字に置き換えて、その大小等で違った処理を行なうので、条件分岐に関してはほとんどの言語[13]で実装されている。そしてその構文にはほとんどで “`if`” が使われている。通常その後に条件を書くので、英語で『もし○○ならば、』という意味になり書き方によっては文章のように直感的に見える[14]。で、`if` 文に関してはほとんどの言語で同じような構文をしてるので、話題になるようなネタはない。

が、“`else`” は言語ごとの微妙な違いがちょっとオモシロイ。`else` は `if` に当てはまらなかった時の処理を記述するもので、

```
if(expression) statement1
else statement2
```

のように書かれる。C や Java の場合 `else` 以下にさらに if を書くことにより何とおりもの分岐を構成できる。つまり、“`else if`” というのは実は、`else` ブロック内の `if` 文のことで特殊な構文ではない[15]。しかし、多くの言語は `if` ブロック下でさらに分岐をさせる時、特殊な構文を用意している。BASIC は `elseif`、Perl は `elsif`、bsh は `elif`[16]、とまあ各言語バラバラだ。そして、`if` 以下の条件分岐にも特殊構文を採用しているのはスクリプト言語に多い。

条件分岐は `if` だけではない。さまざまな言語で `switch` も使われる。これはある変数の大きさによって何通りかの条件分岐をひとつの構文として書くものだ[17]。BASIC の場合、比較演算子も使えることからかなり利用された。

3.4.2 繰り返し

繰り返し文は多くの場合、繰り返しから抜け出すための条件（あるいは、繰り返すための条件）を記述できるようにしているので、条件分岐の 1 つだ。ここで大きく 2 つの繰り返し構文について考えてみる。

GOTO 型

これは、BASIC や Fortran で多く用いられる方法で、起点となる場所になんらかの “label” をもうけ、一定の処理（*`statement`*）をした後、`goto` 文によって label の場所に戻ることによって、

[13] Lisp 系言語等を除く
[14] ただし英語に慣れ親しんだ人にとってだ。
[15] Lisp, Pascal も同様だ。
[16] これは file をさかさまに並べたものだったり。
[17] FORTRAN の `if` は変数の値が正、零または負によって 3 通りに分岐できる。

繰り返し処理を行なうものだ。一般的に、

```
label:
  statement
goto label
```

のように書かれる。BASIC の一部や Fortran は行番号を label して処理する。この方法は、label をつければどこにでも飛ばすことができるため、複雑な処理をするプログラムを比較的簡単に記述できる。

が、goto を多用すると 3ヶ月もたたないうちに自分ですらプログラムが理解できなくなるという罠がまちうけている。特にたくさんの人でひとつのプログラムを書く時は、これは絶対に避けなければならないので、goto はなるべく書かない方がよい。

本物のプログラマ[18]（[4] 参照）を目指すなら別なのだが....。

ブロック型

goto 型の繰り返しの欠点を補うのがブロック型の繰り返しだ。これは一般的に、

```
while(expression){
  statement
}
```

というように、一定の処理（*statement*）をブロック[19]としてひとつの処理とみなし、繰り返しの起点と終点[20]、そして繰り返し後の処理を明確に規定する方法だ。

制約が大きいのでプログラムの構成をあらかじめ考える必要があるが、後からプログラムを見た時に見やすい[21]ので、多くのコンピュータ言語で実装され利用されている。

3.4.3 手続き型、プロセス型

「[2] 計算機プログラムの構造と解釈」によるとプログラム処理には「手続き型」と「プロセス型」の 2 種類がある。

手続き型プログラムとは図 3.1 のフローチャートのように、ある処理をした後別の処理を順番に行ない、条件分岐や繰り返しによって処理の順番を変えたりすることに主眼をおいたプログラムだ。これは単純計算の繰り返し等に向いている。前節で述べた繰り返し処理は、手続き型プログラム固有の問題となる。

一方、プロセス型プログラムとは図 3.2 のように、コンピュータのある状態から別の状態へ遷移を定義し、逐次、別状態へと以降させることに主眼をおいたものだ。次への遷移が定義されていない状態になれば、それによってプログラムの終了となる。再帰関数や論理回路など状態変化に主眼をおくシミュレーションに向いている。

[18] 構造化プログラミングを否定する名著に登場する言葉だ。
[19] C や Perl,Java では中かっこ {} で囲み、Pascal 等では begin と end で囲む。
[20] 終点は異なることがある。
[21] そして美しいプログラムになる

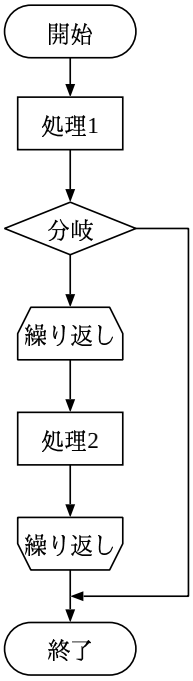


図 3.1: 手続き型

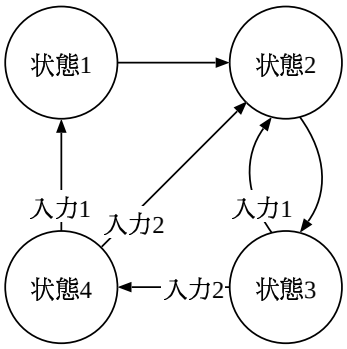


図 3.2: プロセス型

初期の FORTRAN[22]が完全な手続き型で、Scheme が完全なプロセス型であったが、最近のコンピュータ言語にはいずれの要素も含まれる傾向がある。基本的には手続き型で、関数を定義できるようにすることによってプロセス型の言語を実装している。どちらの要素もプログラムには欠かせないのだ。

3.5 プログラムの美しさ

ここまで、プログラムの種類や性質についていろいろ語ってきたが、個人的に思うのはプログラムは美しいことが一番だ。美しいとは、読みやすく、効率がよく、バグがない[23]ということだ。

ここからは独断と偏見になってしまうけど、コンピュータ黎明期の言語である FORTRAN や Lisp は読みにくいという点やバグが多くなりがちになるという点でおせじにも美しいとはいえない。COBOL や BASIC もバグが多く効率もよいとは言えず、なにより見ためにも美しくない。

ここで、私のイチオシの言語が Perl5 だ。Perl4 のイメージのために非常に汚いというイメージ[24]の付きまとう言語であったが、Perl5 になり美しく書くことが可能になった。テキスト処理だけでな

[22]Fortran90 以降は小文字で書くのが正しいが、FORTRAN77 以前は大文字で書く。
[23]最初に語った、不精、短気、傲慢とつながっている。
[24]CGI では関数を利用しない方が効率がよかったので、さらに Perl プログラムは汚いというイメージを強めてしまった。

くバイナリデータの処理も可能で、通常の目的であれば不可能な作業は少ない[25]。スレッドを用いればCで作られたのと同様多数を相手にしたサーバプログラムだってできる。ANSI準拠のCも「1つの作業に1とおりの記述方法」という哲学のおかげでたしかに美しいのだが、拡張性の低さから、単純な作業をいちいち関数で処理したり、重複を避けるための名前の長い関数（Cのライブラリに多くある）は見るに耐えない。それに、入れ子のブロックから一度に何段階も抜ける時、Cでは`goto`を使うか、ややこしいブロック構造にするしかないが、Perlは指定したブロックから抜けることができるので`goto`を使わないすっきりとしたプログラムを記述できる。

Perl5はオブジェクト指向のプログラムを実装することが可能で、関数名をやたら長くしたりせずにすむ。C++やJava,Rubyなどはオブジェクト指向で、美しく書けるようにみえるが、クラス等の修飾子で長々と単語が連なるのは、あまり見ためが良くないし、何がどうなってるのかわかりづらい。それに、書法が限られていて自分好みにすることもできない。Perl5なら自分で`try catch`ブロックを作ることが可能だ。

Perlは遅いって思っている人もいるかもしれないが、それは大きな誤解。変数初期化後にdumpしてとっておけば高速な起動もできるし、一度起動してしまえば文字処理やハッシュの処理などは圧倒的に速い[26]。

これからプログラムを書くならPerl5だ。だまされたと思って試してみよう。きっとその美しさの虜になるに違いない！

[25] サーバプログラムをperlで書けばセキュリティホールも減るだろうに...
[26] それに、実行速度だけでなくプログラムを書くのも速い。

参考文献

[1] Larry Wall, Tom Chiristiansen and Randal L. Schiwartz 共著近藤嘉雪 訳 1997 「Programing Perl」 オライリージャパン

[2] Gerald Jay Sussman, Harold Abelson, Julie Sussman 共著和田英一 訳 2000 「計算機プログラムの構造と解釈」 第二版 ピアソンエデュケーション

[3] 富田豊 大恵俊一郎 真鍋俊彦 黒瀬能 1995 「FORTRAN77 プログラミング」 共立出版

[4] bit 1985 年 4 月号 「本物のプログラマは Pascal を使わない」 共立出版

4 「MML」で作曲廃人～Vol.1.0

01230717 電子情報工学科 2 回生 田中 大義

棚■こんちはー、棚香です（´▽`）ノ
暑いですねー（´ー`）y-~~
何故 11 月なのに暑いのかって？
其れはこの原稿を書いている時期がまだ 9 月だからなのですよ～ヽ（´ー`）ノ
Mew □余計な事云っとらんとさっさと本編入れや !!!（`□´）くわっ＞棚香
ちわっす、Mewkicky（みゅうきき）です☆★☆
棚■うわん、オマエ勝手に出てくるなよヴァカ !!!（#・∇・）つ）;´д`）ぐぶっ
Mew □うわぁん、棚香がいぢめるよぉ～（;ノд`）
って、こんな事してる暇（ページ数）無いんだよヴォケ !!! ガタンッ（ノ`□´）ノ⌒┻━┻）;´д`）げふっ
馬鹿はほっといて早速本編へ♪♪♪

□ Mew です♪、先ず私から（´ー`）ノ
「MML」って云うのは、「Music Macro Language」の略、又は「Music Markup Language」の略なのですよ～♪
んで、其の中身はと云うと、「マシン内の音源を鳴らして、ひとつの曲を演奏するプログラミング言語」って処ですわな（´-`）
原理的には「MIDI シーケンスファイル[1]」に凄く近いんですよ…
まぁ、「MML」の方はプログラミング言語なんで、「コンパイル」してやる必要があるんですが（´ー`）y-~~
此処で注意して欲しい事！、「MML」はコンパイラごとに書式が違うんです !!!
今回の私等の「MML」のソースの書式は「テキスト音楽サクラ[2]」の物です（´▽`）ノ

```
//_初期設定 [ META EVENT ]
TrackName     = {"天使降臨マップ曲"}     //曲名
Copyright     = {"KITCC 天使降臨プロジェクト"}     //著作権表示
MetaText      = {"マップ曲"}     //説明・コメントなど
Tempo         = 210;     //テンポ設定
TimeSignature = 4,4;     //拍子記号の設定→ 2,4 は 2/4 拍子を表す
//_初期設定 [ System Setting ]
System.MeasureShift(0)  //小節数のシフト設定
System.TimeBase(96)     //四分音符分解能
System.KeyFlag+()       //調号の設定→ドとファに＃をつける
ときは、+(cf) と設定
System.vAdd(10)         //v++,v--で変化する値
System.qAdd(10)         //q++,q--で変化する値


TR=1 CH=1;
```

[1]インターネット上でよく「MIDI」と呼ばれている「.mid」形式ファイルの事、但し「MIDI（Musical Instruments Digital Interface）」自体は音楽機材同士、又は音楽機材とパソコン等をつなぐ為の規格であり、本来「.mid」形式ファイルを指すものではない。
[2]http://www.text2music.com/ で配布されている「MML」の開発＆コンパイラ環境、フリーウェアコンテストで賞を取っている。

```
Voice=85;

l8 v120 [3 [r2rdef g4f4e4d4 c^^^^<a#>c4 <a1> r2rdef g4f4e4c4
 a^^^^gf4 g1]
l8 v120 a^^^^a#>c4 <g4f4e4f4 g^^^^fe4 fed^^^^^ a^^^^a#>c4
 <g4f4g4>c4 c1 r1<
l8 v120 a^^^^a#>c4 <g4f4e4f4 g^^^^fe4 fed^^^^^ a^^^^a#>c4
 <g4f4e4c4 d1 r4rdrdr4]


TR=2 CH=2;
Voice=35;

l8 o3 v90 [3 [[d4dd4dd4] c4cc4cc4 <a4aa4aa4> d4dd4dd4 d4df4fd4
 g4gf4fg4 >c4c<a#4a#a4]
l8 o3 v90 <[4 a#r]> [4 cr] <[4 ar]> [4 dr] <[4 a#r]> [4 cr]
 [8 dr]
l8 o3 v90 <[4 a#r]> [4 cr] <[4 ar]> [4 dr] <[4 a#r]> [4 cr]
 [4 dr] d4rdrdr4]


TR=3 CH=3;
Voice=28;

l8 v100 [3 [4 [d4fa4fd4] c4eg4ec4 <a4>ce4c<a4>]
l8 v100 <'a#>df<'1> 'ceg'1 <'a>ce<'1> 'df>a<'1 <'a#>df<'1>
 'ceg'1 'df>a<'1 'd#f>a<'1
l8 v100 <'a#>df<'1> 'ceg'1 <'a>ce<'1> 'df>a<'1 <'a#>df<'1>
 'ceg'1 'df>a<'1 'df>a<'4r'df>a<'r'df>a<'r4]


$b = {n36,} // bass drum
$s = {n38,} // snare
$h = {n42,} // HiHat closed
$m = {n44,} // HiHat open
$c = {n49,} // Clash


TR=10 CH=10;
Rythm{

l8 v100 [8 h4'hs'4] [c4'hs'4 [7 h4'hs'4]] c4'hs'4 [5 h4'hs'4]
 m4'ms'4'ms'4'ms'4
l8 v100 [15 'cs'4'ms'4'ms'4'ms'4] c4rc4c4r
l8 v100 [[3 c4'hs'4 [7 h4'hs'4]] c4'hs'4 [5 h4'hs'4]
 m4'ms'4'ms'4'ms'4
l8 v100 [15 'cs'4'ms'4'ms'4'ms'4] c4rc4c4r]

}


TR=11 CH=10;
Rythm{
```

```
l8 v110 [3 [4 [3 br4brbr4] bbrbrbr4]
l8 v110 [60 rb] rbbrbrbr]
}
```

□って、いきなりソース見せてもわかりませんよね（^-^;）
皆さんは「ドレミファソラシド」ぐらい解りますよね？(いやマジで…)
此れを「ｃｄｅｆｇａｂｃ」と表すと聞いた事があるかな???

```
TR=2 CH=2;
Voice=35;

l8 o3 v90 [3 [[d4dd4dd4] c4cc4cc4 <a4aa4aa4> d4dd4dd4 d4df4fd4
 g4gf4fg4 >c4c<a#4a#a4]
l8 o3 v90 <[4 a#r]> [4 cr] <[4 ar]> [4 dr] <[4 a#r]> [4 cr]
 [8 dr]
l8 o3 v90 <[4 a#r]> [4 cr] <[4 ar]> [4 dr] <[4 a#r]> [4 cr]
 [4 dr] d4rdrdr4]
```

□変な記号がいっぱいついてますが、「ａｂｃ～」だけ見てると、何と無く譜面っぽく見えてきませんか???
この「ａｂｃ～」が音階を表してるんです（´ー`）ノ
それでは、この変な記号について少しずつ見てみましょうか？

ａ ｂ ｃ ～　音階を表しています、「c：ド、d：レ、e：ミ、f：ファ、g：ソ、a：ラ、b：シ」という感じ。
ｒ　休符を表しています、無音を表す音階記号だと考えれば良いです。
ｇ4 ｒ2 ｃ1 ～　（音階記号の後ろに付いている数値）：何分音符かを表しています。
~　タイ記号、この記号の数だけ音が伸びます、コンパイラによっては使えないこともあります。
ｌ　音階記号の後ろに数値を書かなかった場合の段落全体の音長が何分音符かを表します。
ｏ　オクターヴ[3]、つまり音の高さを表します、此処では段落全体のオクターヴを表します。
ｖ　ヴェロシティ、音の強さです、アタック感と云うより音量が変化します。
ｑ　ゲートタイム、この指定の後の音階記号の音長に対する発音時間を、q（百分率の数値）で指定します。
ｔ　タイミング、この指定の直後の音階記号の発音タイミングをずらします、t（数値）で指定します。
 # +　シャープ記号（＃）です、音階記号の後ろに付けることで其の音を半音上げます。
-　フラット記号（♭）です、音階記号の後ろに付けることで其の音を半音下げます。
[ ]　繰り返しを指定します、繰り返し演奏回数も指定できます、デフォルトは 2 回演奏。
:　[ ] による繰り返しから脱出します、脱出タイミングは繰り返し演奏の最後の回です。
>　この後の音階全部を 1 オクターヴ上げ[4]ます。
<　この後の音階全部を 1 オクターヴ下げ[5]ます。
’ ’　此れで囲まれた音は同時に鳴らされます。

□後は「初期設定」さえ書けば、「MML」は完成です（´ー`）ノ
「初期設定」はコンパイラによって異なりますが、基本は 1 ページ目のような感じで書かれます。
でもそれだけじゃ、ほぼ単音のみの非常につまんない曲になってしまいますな（´д`;）うへぇ
なので、もっと曲に奥行きを出す為にトラック数を増やしましょうヽ(´ー`)ノ
記述はコンパイラによって違いますが、手順は大体一緒です。

トラック番号の指定　音階記号の集合、此処では「TR=」と書かれています、トラック数は最大 16、トラック 10 はデフォルトでリズムパートとなっています。
チャンネルの指定　音色の集合（但し 1 チャンネル 1 音色しか割り当てられません）、此処では「CH=」と書かれています、チャンネル数は最大 16、10 チャンネルを指定するとリズムパートになります。
音色の設定　チャンネルに音色を割り当てます、此処では「Voice=」と書かれています、音源の音番地から好みの音を割り当てます。
タイミング設定　トラック全体の演奏開始タイミングを設定します。

□此れでトラックの割り振りも出来ましたヽ(´▽`)ノ
因みに「トラック 1」は「シタール[6]とディストーションギター[7]を足したようなシンセリード」、「トラック 2」

[3] ドを基準とした場合、次に来るドまでの間を 1 オクターヴという。
[4] 6 音上げ、ドを基準とした場合、一段階高いドが 1 オクターヴ上
[5] 6 音上げ、ドを基準とした場合、一段階低いドが 1 オクターヴ下
[6] インドの民族弦楽器、ギターなどでは嫌われる弦とフレットが触れて出るビビリ音を意図的に作り出したトーンが逆に美しく響く。
[7] 出力される音の波形の頭部分が潰れた状態になることで出る歪んだ音、硬質な歪みを特にディストーションと呼ぶ。

は「ピック弾きベース」、「トラック 3」は「クリーントーンギター[8]」になってます
後はリズムパートを残すのみ、私は疲れたので棚香に交代です（´ー`）y-~~

■ Mew のやつ〜、わたしが言いたいところ殆ど持っていきやがった（`□´）くわっ
もう残り少ないじゃんかぁ（T д⊂）くそぅ
んじゃ、気を取り直していきます（#´-`）ノ

```
$b = {n36,} // bass drum
$s = {n38,} // snare
$h = {n42,} // HiHat closed
$m = {n44,} // HiHat open
$c = {n49,} // Clash

TR=10 CH=10;
Rythm{

l8 v100 [8 h4’hs’4] [c4’hs’4 [7 h4’hs’4]] c4’hs’4 [5 h4’hs’4]
 m4’ms’4’ms’4’ms’4
l8 v100 [15 ’cs’4’ms’4’ms’4’ms’4] c4rc4c4r
l8 v100 [[3 c4’hs’4 [7 h4’hs’4]] c4’hs’4 [5 h4’hs’4]
 m4’ms’4’ms’4’ms’4
l8 v100 [15 ’cs’4’ms’4’ms’4’ms’4] c4rc4c4r]
}
```

■「$ = {n ,}」って云うのは、音源からリズムパート用の音を呼び出す為のマクロだよ（´ー`）ノ
上に書いてある「b s h m c」の部分は、別の好きな半角小文字英字でも代用できるので、解り易い物を設定しよう。
「n ,」の間には音源のリズムパート用の音の番地が入ります、此れが指定されてはじめてリズムパートが鳴るのですよ〜♪
因みにリズムパートに使ってる音は上から「バスドラム[9]」「スネアドラム[10]」「クローズハイハット」「オープンハイハット」[11]「クラッシュシンバル[12]」

■譜面の記述方法は音階の記述方法とほぼ同じです（´-`）
「a b c 〜」が「b s h m c」に換わっただけだね…
後は譜面部分を「Rythm{ }」で囲ってやるだけ、Wow！、簡単やね？
但し「Rythm{ }」を忘れるとリズムパートは鳴りません、注意！(`ー´)
また、コンパイラによって「Rythm{ }」部分が異なる場合があるから其処はコンパイラのマニュアルに従うようにヽ(´ー`)ノ

■後はコンパイルするだけやね、最初に言ってたように、「MML」はコンパイラごとに記述が違うから、其処についてはコンパイラのマニュアルに従って下さいな。

---

[8] ディストーションギターとは対照的に歪みの無いトーンのギター、一時期隆盛を誇ったゴシックロックの類でよく使われた、代表的なのは LUNA SEA の INORAN。
[9] ドラムセットで足元に横たえてセットされている口径の大きいタイコ、ペダルを踏んで鳴らす、「ドン！」という低い音が出る
[10] ドラムセットで口径 14 インチ程度で、胴の浅めなタイコ、「タン！」というアタック感がある音が出る。
[11] ハイハット、シンバルを 2 枚重ねにして、ペダルを使って 2 枚のシンバル同士を接させたり、放したりする、閉めた状態では「チチチ」という感じの音。
[12] 比較的大口径のシンバル、パシャーンという音が出る。

でも、基本的な部分は同じだから、そんなにコンパイラごとに位置から勉強しなおす必要は殆ど無いです。
また、コンパイラによっては「MML」から「MIDI シーケンスファイル」を生成するものもありますから。
これらをマスターすれば自分のホームページとかで手軽に音楽が流せるようになりますな（´ー`）y-~~

棚■取り敢えず、おわったぁ（´д`;）
因みに上の例で引用した曲[13]は、「KITCC[14]」内で作成している P/ECE[15]用ゲームの BGM[16]だよ（´ー`）ノ
其のゲームをテンプレートにオンラインゲーム[17]を作るだなんて計画があるんだよね…
Mew □えらい壮大な計画やなぁ…
何時完成するんやろうか???
って云うか、アンタ、まだ大事な事忘れてないか???
棚■知りませんな（´ー`）y-~~
Mew □確信犯かよ!!!、何か云いやがれ（°Д°）!!!
棚■だってぇ、ページ数が足りないんだもぉん♪
Mew □うへぇ（´д`;)、どうするのよ、「関数」「マクロ」「制御構造」[18]とかは!?
棚■次回に回せばぁ???、だって残りのページ数じゃ絶対足りないよぉ（´ー`）ノ
其の辺話すと長いし、初めての人がとっつきにくいじゃん???
という事で、次回「「MML」で作曲廃人〜Vol.2.0」をお楽しみに（´▽`）ノ~~
Mew □って云うか、また対談形式[19]でやるんか???
棚■さぁねぇ、┐（´へ`;）┌

[13]実際に「テキスト音楽サクラ」を使って、この曲をコンパイルしてみると、実は結構「LUNA SEA」っぽい曲になってる。
[14]京都工芸繊維大学コンピュータ部の略称、kitcc.org を取得している。
[15]アクアプラス社の発表した、C 言語のプログラミングで比較的手軽にゲーム開発が出来る携帯ゲーム端末、この端末用の「MML」もまた微妙に記述が異なる。
[16]ゲームの BGM は、普通に曲を作るより余計に気を使う、其処が難しいんです。
[17]世界中のプレイヤーがネットワークを通して同じゲーム世界を共有して遊ぶゲーム、執筆当時は GRAVITY 社の「RAGNAROK ONLINE」が盛況だった。
[18]「MML」の記述を簡潔に出来る他、特殊な操作が出来る、詳しくは次号にて。
[19]別称エスキセ形式（笑）

編集後記

去年にひきつづき編集委員を勤めました山本です。今年も LaTeX 2ε を利用しての Lime 作成となりました。去年は学籍番号順に記事を並べていたので、内容がごちゃごちゃしてしまいましたが、今年は分類わけして見やすいようにしました。

今年はどれもよく調査されていてしっかりとしている記事だと思います。ハードウェア関連の記事は少ないですが、とても興味深い内容でどんどん読めました。最近の流行もあってネットワーク関連の記事が多いですが、どの記事も実用的で初心者から上級者まで読む価値のあるものだと思います。私はソフトウェアで記事を書きましたが越本部長にはかないそうにないです。(^-^; 田中の MML の記事はコンピュータ部内で数少ないマルチメディアに強い部員として弧軍奮闘してくれていると思います。

例年同様土壇場になって完成させるという人がいたりして、編集にはやはり苦労してしまいました...。しかし、最後にはみんな原稿が集まりほっとしています。来年こそは夏休みまでの完成を目指したいものです。

来年は研究室に入り少し忙しくなるので、この Lime の製作の方法を後輩が作れるようにしていきたいです。

平成 14 年 10 月 20 日 編集担当 山本大介