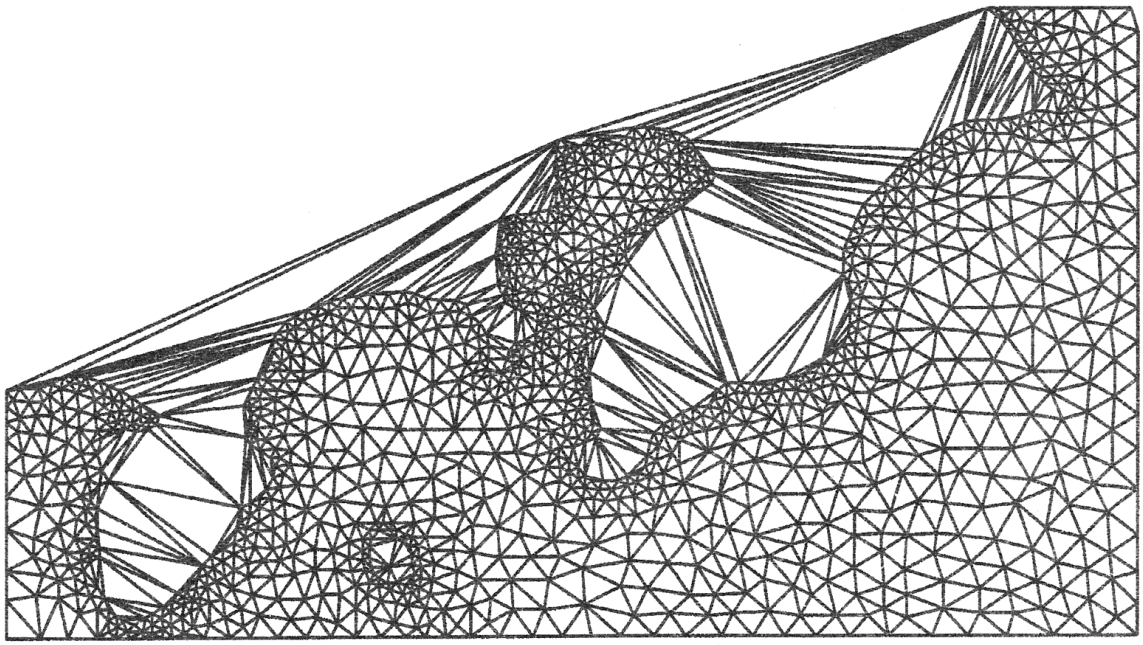
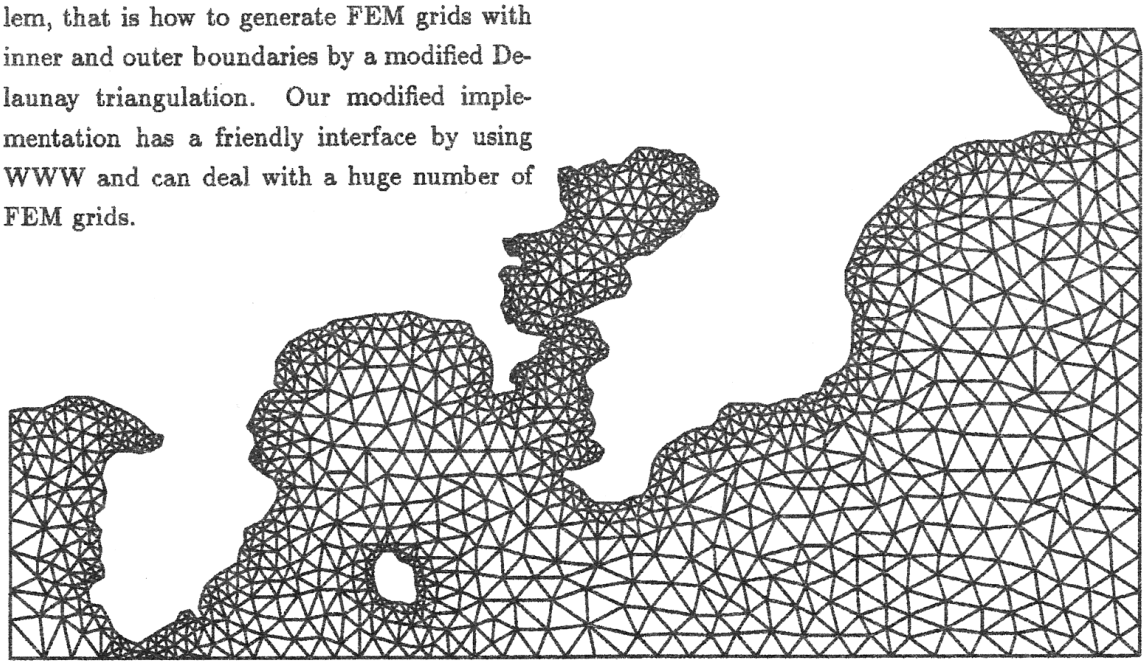
百万石

Microcomputer Making Association

1996-11-15



The Delaunay triangulation is a well-known method to generate FEM grids. However, a simple Delaunay triangulation ignores FEM domains' boundaries. So, we solve the problem, that is how to generate FEM grids with inner and outer boundaries by a modified Delaunay triangulation. Our modified implementation has a friendly interface by using WWW and can deal with a huge number of FEM grids.



1996年 調布祭シーズン 百萬石

目次

電気通信大学MMA

Part I
ネットワーク編

学外向けメール配送の複数経路化

楯岡 孝道 (tate@spa.is.uec.ac.jp)

Abstract

MMA はプライベートアドレスを用いて IP ネットワークと接続しているため、学外との直接通信が行えない。そのため、学外とのメール交換には中継ホストを経由する必要がある。現在は単一の中継ホストを経由しているため、この中継ホストとの接続が失なわれるとメールの交換ができなくなる。本稿では特に MMA から学外へのメールを中継する中継ホストを複数にし、メール配送を複数経路化することにより、メールの可用性を高める提案を行う。

1 はじめに

MMA は学内の IP ネットワークとの接続に RFC1918 [1] で規定されているプライベートアドレスを用いている [2]。プライベートアドレスは組織内のみで有効なアドレスであるので、MMA と学外との IP レベルでの直接通信は行なえない。これに対処するために、アプリケーションレベルで中継を行なうアプリケーションゲートウェイが存在し、電子メールに関しては特定のホストを中継して学外とのメール交換を行なっている。

しかし、現在 MMA が利用しているメール中継ホストは `ns.uec.ac.jp` の一台のみであり、このホストとの接続性が失なわれると、学外に対する一切のメールの送受信が不能になる。これに対処する方法として、学外から MMA に対するメールに関しては、`*.club.uec.ac.jp` に対する DNS の MX レコードを複数設定することにより複数経路化が可能になることが知られている [3]。

逆に MMA から学外に対するメール中継ホストは、MTA である `sendmail` の設定ファイル `sendmail.cf` に静的に記述されており、一箇所に固定されている。本稿では学外に対する中継ホストの定義にも DNS の MX レコードを用いることにより、学外に対するメール配送の複数経路化も実現する方法を提案する。

2 現在の配送規則

現在の MMA におけるメールの配送規則は以下のようになっている。

1. メールの宛先が学内であり、そのアドレスに対する private ドメインが存在すれば private ドメインの MX レコードに従ってメールサーバに送信する。
   例: tree@fan.club.uec.ac.jp 宛
2. メールの宛先が学内であり、そのアドレスに対する private ドメインのメールサーバが存在しなければそのまま MX レコードに従ってメールサーバに送信する。
   例: tate@smiley.cs.uec.ac.jp 宛
3. メールの宛先が学外ならば、中継ホストである `ns.uec.ac.jp` に送信する。
   例: tate@wide.ad.jp 宛

ここで問題になるのは、3 の規則である。中継ホストを静的に指定しているため、この中継ホストとの接続が失なわれると学外に対するメールの配送が停止してしまう。また、中継ホストが変化した場合には、全ての計算機上の `sendmail.cf` の記述を変更しなければならない。

3 MX による複数経路化

`sendmail.cf` で記述するメール中継ホストには、DNS の MX レコードを指定することができる。この場合 `sendmail` は指定された MX レコードを DNS から取得し、それによって配送先を決定する。

そこで、「学外向けメール中継ホスト」を意味する MX レコードとして `outmx.mma.club.uec.ac.jp` を定義し、各計算機の `sendmail.cf` の中ではこの MX レコードを静的配送先として定義する。このように定義することにより、学外

向け配送ルールも DNS の MX レコードで制御できるようになる。MX レコードは優先度を付けて複数定義できるため、バックアップ経路としての指定なども容易である。

試験的にバックアップ経路を設定したところ、主経路の `sendmail` を停止するとバックアップ経路を通過してメールが配送されることを確認できた。

4 問題点

今回提案した方法は、筆者が自宅で数ヶ月にわたって利用したこともあり、技術的な問題点は殆どない。むしろ非技術的な問題が多い。以下に問題点を記す。

中継ホストを複数用意できるか MMA 所有の計算機は全てプライベートアドレスを利用しているため、中継ホストとしては学内のネットワーク管理グループの計算機を利用させてもらっている。そのため、複数経路化のために中継ホストを増やすには交渉が必要である。

配送の非対称性 学外から MMA に向けての中継ホストには、private ドメインに対する特殊な設定が必要 [3] だが、MMA から学外に向けての中継ホストには特別な設定は不要なので、問題の起きないようなホストを利用してしまうことは可能である。

しかしその場合、メールの転送方向によって配送経路が異なることになり、学外からのメールは到着しないが、学外へのメールは正常に配送されるような現象が生じる。

5 今後の予定

中継ホストが複数用意できなくても、今回提案した方法により中継ホストが変化した場合などに対処しやすくなる。よって、今後学外向きの配送ルールとして今回提案した方法を利用していく予定である。

参考文献

[1] Y. Rekhter, R. Moskowitz, D. Karrenberg, G. de Groot, E. Lear, “Address Allocation for Private Internets,” RFC1918, 1996.
`ftp://ftp.uec.ac.jp/pub/wwfs/RFC/rfc1918.txt`

[2] 楯岡 孝道, “Private Network Address に於ける DNS 環境の構築,” 1995 年 新入生歓迎シーズン 百万石, pp.18-20, MMA, 1995.
`http://www.uec.ac.jp:8081/club/mma/mma/h_9504_dns.html`

[3] 楯岡 孝道, “プライベートネットワークにおけるメール配送について,” 1995 年 調布祭 百万石, pp.5-10, MMA, 1995.
`http://www.uec.ac.jp:8081/club/mma/mma/h_9511_mail.html`

rfinger のひみつ

しみずりょう

finger

finger は、ネットワーク上のマシンにログインしているユーザの情報を表示するためのものです。その定義と詳細は RFC742 に記述されています。

これらを利用すれば、ユーザ名、実名 (`/etc/passwd` に記述されるもの)、仮想端末名、アイドル時間、ログイン時間等と、かなり有用な情報を得ることができます。

MMA でも finger サーバが、特に動かさない理由もないので稼働しています。これはリモートから部室、もしくは、部室のマシンを、利用しているユーザを知るのにとても有用です。

が、学外から MMA のマシンに finger をかけた際[1]、そのパケットは細い回線を通っていくため、かなり待たされます。ただでさえ遅いのに、

```
% finger @beat.mma.club.uec.ac.jp@raven.cc.uec.ac.jp
[raven.cc.uec.ac.jp]
[beat.mma.club.uec.ac.jp.private]
No one logged on.
% finger @gont.mma.club.uec.ac.jp@raven.cc.uec.ac.jp
[raven.cc.uec.ac.jp]
[gont.mma.club.uec.ac.jp.private]
No one logged on.
% finger @flop.mma.club.uec.ac.jp@raven.cc.uec.ac.jp
[raven.cc.uec.ac.jp]
[flop.mma.club.uec.ac.jp.private]
No one logged on.
```

などと、3 つのホストを調べたりすると、それにかかる時間が 1 分以上なんてこともしばしばです。今後マシンが増えることを予想すると、すべてのマシンに finger するのに十数分... こんなバカげた状況を回避すべく、MMA のマシンに GNU finger を install しました。

GNU finger

GNU finger はこうした問題を解決してくれるものです。

finger サーバが、その他のマシンの finger 情報を常に集めていて、外からサーバに finger をかけると、集めている finger 情報を一度にすべて表示してくれるというものです。

つまり、MMA で finger サーバを一台立ちあげておくと、そのマシンに finger するだけで、MMA のマシンすべての finger 情報を得ることができるのです。

がしかし、問題点もあります。

サーバは、指定された時間間隔で他の全てのマシンの finger 情報を集めるという方式をとっているため、ポーリング時間が長いと、finger 情報は多少タイムラグを持つものとなり、ポーリング時間が短いと、ほとんどタイムラグのない情報が得られるものの、ネットワーク的な負荷が大きくなってしまいます。

[1] 学外のマシンからは MMA のマシンは直接見れないのですが、fingerd のリレー機構を使って見ることができます。

dynamic recursive finger daemon

そこで、finger された時に「動的に」その他のマシンに finger をかけて結果を返す daemon を作れば、これらの問題は解決できます。finger の反応が遅くなるという問題もありますが、MMA のようにホスト数がそれほど多くない環境では大きな問題にはならないと思います。

さらに、finger 情報をある程度の時間キャッシングすれば、短い時間に連続して finger されても、その都度他のホストに finger しにいく、という事態は避けられます。

設計と実装

ということで perl でサクサクと書きました。構想 1 年、作成時間は 2 時間です。

動作は以下の通りです。

- inetd によって、fingerd ポート (79/tcp) のサーバとして起動される。
- サーバにリクエストが来たら、自分の監視すべきクライアントの情報を提示する。

基本はこれだけです。クライアント情報の獲得について以下に説明します。

クライアント情報は、基本的には rfinger サーバがクライアントに対して finger して獲得します。

この際、ファイルシステムを利用してキュッシュ機構を実現させることもできます。クライアントを finger した結果を rfinger のワークディレクトリに書き込んでおき、次に rfinger サーバに対してリクエストがあった場合に、もし前回のリクエストから長い時間が経っていなければ、ファイルシステムにキャッシュしている内容をそのまま提示します。

以下、rfinger のワークディレクトリを`$RFINGER_DIR`、クライアントのホスト名が`$HOSTNAME`、そしてキャッシュの有効時間が`$EXPIRE`とします。

クライアント情報の取得シーケンス

- クライアント名のリストを、`$RFINGER_DIR/clients` ファイルから得る
- `$RFINGER_DIR/cache/$HOSTNAME` が存在し、その作成時間が現在の時間よりも `$EXPIRE` 秒以前のものであれば、クライアントに finger して、その結果を`$FGINER_DIR/cache/$HOSTNAME`に格納する。
- `$RFINGER_DIR/cache/$HOSTNAME` を表示する。

しかし、これだけでは問題がかなりあります。

1. クライアントの fingerd が down していた場合に、処理が止まってしまう
2. クライアントの finger サーバが rfinger であった場合に、処理がループしてしまう

2. に関しては、そういう設定にするほうが悪いと言えなくもないですが、どのサーバに finger しても、すべてのマシンの finger 情報を得られるというふうにもしたいところです。

そこで、クライアントの finger 情報を得るのに、GNU finger の cfinger を利用することにします。cfinger は、finger のサブセットのようなもので、一定のフォーマットで finger と同程度の情報を返す daemon です。

まず、cfinger に問い合わせをし、もし cfinger があればそこから情報を得る。cfinger が動いてなければ、ふつうに fingerdaemon に問い合わせをする、とすれば ok です。

そして、finger サーバに rfinger を使う場合は、GNU finger の cfinger も同時に使わなければならない、とします。こうすれば、finger がループしてしまうことはなくなるでしょう。

1. に関しては、いい解決策が思いつきません。とりあえず、タイムアウトを 1〜2 秒とかにして様子を見ることにします。

が、これだとクライアントが 10ヶあるとして、すべてが down していた場合に 10〜20 秒もかかってしまいます。

そこで、クライアントの finger 情報を得るのには、いっきにクライアントの数だけ fork して、いっせいに finger しにいくことにします。

結果はファイルシステムに書かれるので、チャイルドプロセスが全て終了してからファイルシステムを介して結果を得、表示すれば ok でしょう。

test

とりあえず調布祭期間中に MMA ブースでテストしてみることにします。

注) rfinger は、recursive finger であって、ryo's finger ではありません。`:P`

自動応答機構 ARGO コマンド処理システム

楯岡 孝道 (tree@mma.club.uec.ac.jp)

Abstract

共同作業支援システム ARGO は、電子メールを用いて作業者間の意志疎通と情報の共有を実現する機構である。本稿では、ARGO システム中の、利用者からの命令を処理する ARGO コマンド処理システムに関して述べる。

1 はじめに

筆者はこれまでに SMD (Simple Mail Distributor) と呼ぶ、参加者へのメールの配送のみを行なう、単純なメーリングリストサーバを作成、利用して、桜井智メーリングリスト [3] を始めとする複数の ML (Mailing List) を運用してきた。

しかし、ML への参加人数が増えるに従い、メール配送アドレスの追加や削除の手間が無視できなくなり、これらの自動化が求まれるようになった。また、参加者間で共同作業を行なう際に、作業に必要な情報を保存しておく場が求まれるようになった。

ARGO (Automatic Reliable Group Work Operator) システム [1, 2] はこのような問題を解決すべく設計、実装されたメーリングリストサーバである。ARGO では、メール配送アドレスの追加や削除、メールやファイルの保存や検索等の共同作業に必要な操作を、ほとんど全てメールによる命令 (コマンドメール) で行なうことを可能にした。

2 ARGO プロトタイプと新配送系

筆者はこれまでにコマンドメール処理システムとして ARGO プロトタイプ [1] を実装し、さらに SMD に代わるメール配送の為の機構として新配送系 [2] を設計、実装した。

ARGO プロトタイプの試験運用によって、ARGO プロトタイプの以下のような問題が明らかになった。

- 配送方法が限られており、利用者の要望に応えきれない
- 命令形態が複雑であり、利用者が理解しにくい
- 管理者のみが利用する特権命令が存在しない

これらの問題のうち、配送方法に関しては新配送系によって解決された。

しかし、新配送系の用いる配送リストデータベースは、ARGO プロトタイプの用いているものと異なる形式であるため、そのままでは共に利用できない。そこで、上記の問題を解決し、新配送系と共に利用可能なコマンドメール処理システムを設計、実装する必要があった。

3 ARGO コマンド処理システムの設計

ARGO コマンド処理システムは以下の目標を達成すべく設計された。

1. 命令形態は極力単純にし、一般利用者が理解しやすいものとする
2. 可能な限り制限を設けない
3. 不正な入力があっても、それを無視できる
4. 細かな仕様の変更や、機能の限定を可能とする
5. 管理者は必要に応じて利用者のコマンド実行を監視できる
6. 管理者のみが利用可能な特権命令を用意し、一般利用者からこれを保護する

これらの目標を達成すべく、ARGO コマンド処理システムは以下のような特徴を持つ。

ユーザに優しいインターフェース

ARGO システムの利用者は計算機の利用に慣れているとは限らず、複雑なコマンドの動作や意味を理解できないことがある。

ARGO プロトタイプでは、ファイルを保持するライブラリ機能において、更新されるファイルの一意性を保証するために、マジックナンバーと呼ばれる番号を利用していた。しかし、このためにコマンド形態が複雑になり、理解しにくいものとなっていた。そこで今回の設計では、ファイルが更新される度にファイルを識別するファイル番号を新たに割り当てることにより、一意性を保証しつつ、コマンド形態を単純にした。

また、コマンドメール中で入力されたコマンドと、それを処理した結果を併記して、それを応答メールとして返信することにより、ユーザがどのコマンドに対してどのような結果が得られたかを理解しやすいようにした。

制限の廃止

利用者の利用している環境はさまざまであり、一般に受信可能なメールの大きさ等を判断することはできない。そこで、今回の設計では ARGO 側ではコマンド長や応答メールの大きさなどに制限を加えなかった。

ただし、ARGO が動作している計算機の能力を越えた要求に対しては、処理を拒否する。

不正入力への耐性

一般の利用者は誤ったコマンドを与えたり、コマンドメールにシグネチャと呼ばれる署名を付加するなど、さまざまな不正入力を行なう。特に誤った命令として漢字が含まれていた場合にも、応答メールのキャラクタセットが正しく保存される必要がある。

今回の設計では、ユーザの意図しない動作をしないように、コマンドの識別を厳密に行なうようにした。また、コマンドとして漢字が含まれていた場合には ISO-2022-JP のエスケープシーケンスを削除することで、キャラクタセットが変化するとを防いでいる。

設定による動作の変更

ML によっては、特定のコマンドを無効にしたり、特定のコマンドを実行した際に特別な処理をする必要がある。

これらの要求に応えるため、設定ファイルを用意し、これによって標準の動作を上書きすることによって、ML によって異なる動作を可能にした。

コマンド監視機構

ML の管理者は ML のファイル管理機構が正しく利用されているかどうか、不正に利用している者がいないかどうか監視する必要がある。

これに対処するため、コマンドメール及び、応答メールの一部を複製し、管理者に送信する機能を持つ。

特権命令

ML の管理者は時に、利用者の強制的な利用者リストからの削除や、メール配信方法の変更等を行なう必要がある。しかし、これらの命令を管理者以外が利用できてはならない。

これに対処するため、特権モードを用意した。特権モードでは利用者リストの操作や、任意の利用者の権限でのコマンド実行が可能である。特権モードへの遷移には、メールの送信者アドレス及びパスワードが確認され、認められた場合のみ遷移が許可される。送信者アドレスを偽造した場合には特権命令の実行結果が、偽造された、正しい管理者のアドレスに返信されるため、管理者は偽造されたコマンドの実行を知ることができる。

4　利用可能なコマンド

ARGO コマンド処理システムで利用可能なコマンドは以下の通りである。

guide ML の紹介文を取得する。このコマンドと subscribe のみは、登録された利用者以外にも使用可能である。

subscribe ML 利用者としての登録を行なう。

help コマンドの使い方などの解説文章を取得する。

mode ML 宛メールの配送方法を変更する。

member 登録されている利用者の一覧を得る。

get ML に過去に流されたメールのうち、任意のものを取得する。

list ML に過去に流されたメールの一覧を取得する。同時に送信者アドレス及び題名の検索も可能である。

libget ライブラリに登録されたファイルのうち、任意のものを取得する。

libput ライブラリに新たなファイルを登録する。

libupdate ライブラリに登録されているファイルを更新する。

libappend ライブラリに登録されているファイルに追加を行なう。

libdelete ライブラリに登録されているファイルを消去する。これは登録した本人のみが可能である。

liblist ライブラリに登録されているファイルの一覧を取得する。同時に送信者アドレス及び題名の検索も可能である。

su 特権命令モードに遷移する。

5 ARGO コマンド処理システムの実装

ARGO コマンド処理システムは、ARGO システムの他の部分と同様に perl スクリプトで記述されている。コマンド処理システムはコマンドの解釈を行なう `command.pl` と、処理結果を MIME multipart 形式 [4] に整形して送信する `compack.pl` から構成される。

ARGO システムにおいてはコマンドメールは ML 宛メールとは別のメールアドレスを持ち、そのメールアドレス宛のメールは全て `command.pl` に入力される。ここでメールヘッダの解析を行ない、解析によって得られた送信者アドレスから、コマンドが登録された利用者からのものか否かを判断する。その後、メールの Subject: (題名) 及び本文をコマンド列とみなして実行する。

コマンドに対する応答メールの内容は、入力されたコマンド列とそれに対する応答メッセージ列 (Command Output) と、コマンドによって返信されるべきメールメッセージやファイルからなる。これらは別々の一時ファイルとして作成され、最後に `compack.pl` により MIME multipart 形式でコマンド発行者に送信される。

管理者がコマンド実行結果の監視を指定した場合には、Command Output のみが管理者に送られる。これは、応答メールの内容を全て送ると管理者宛のメールが巨大になり、管理者の負担が大きくなるためである。

設定ファイルは起動時の引数で指定し、スクリプトの初期化終了後に perl スクリプトとして読み込む。これにより、コマンド名とコマンド処理関数の対応を表わす配列を含む、任意の変数を変更することが可能になっている。

現在の実装では、`command.pl` が約 1600 行、`compack.pl` が約 100 行程度の規模である。

5.1 システムの運用上の問題点

現在、2 つの ML で ARGO システムを運用し、更に 1 つの ML で ARGO システムのライブラリ機能のみを利用している。最も長く利用している ML では、半年近く ARGO の運用を続けている。

これまでの運用によって明らかになった問題点は以下の通りである。

コマンドを間違える利用者 他のメーリングリストサーバとコマンド体系が異なるためか、コマンドを間違える利用者が目立った。しかし、正しいコマンドが一つも実行されなかった場合にはコマンドの使い方を記したファイルを返信する設定にした結果、ほぼ全員が直後に正しいコマンドを再送してきた。

全角文字のコマンド ISO-2022-JP コードセットの全角文字を用いてコマンドを記述する利用者が存在した。最近のメーラでは全角文字と、us-ascii に含まれる半角文字の区別がつきにくいものが増えているためだと考えられる。これに関してはエラーとして処理している。

応答メッセージの不備 一部の応答メッセージは英語で記述されているが、利用者の中にはこれを理解してくれない者も存在する。このため、特にエラーメッセージに関しては日本語化が必要であると考えられる。

エラー処理の不備 初期の実装では一時ファイル保持領域が不足した際のエラー処理が不十分で、その後のコマンド実行に障害が発生したが、現在の実装では対処されている。

無理なコマンド要求 自らの受信能力を考慮せずにコマンドを発行し、応答メールを受信しきれない利用者が存在した。

応答メールの大きさの制御 現在は利用者が応答メールの大きさを判断する材料が不足しており、利用者が経験によって発行するコマンドを変化させているのが現状である。また、応答を分割して送信する機能などが無いため、ライブラリ等に巨大なファイルが登録されても、それを取得できない利用者が生じてしまう。

処理速度 ML メールのアーカイブ、及びライブラリに保存されたファイルは、全て RFC822 に準拠した形式で 1 メールが 1 ファイルとして記録されている。このため、記録されているメール量が増えると処理速度が低下してしまう。実際には、Pentium 100MHz の IBM-PC 互換機に PCI バスの SCSI I/F と SCSI HDD を接続し、OS として BSD/OS を動作させている環境で、4400 通のメールがアーカイブされている中から題名と送信者アドレスを指定して 6 通のメールのリストを生成するのに、約 1 分間を要する。

処理能力の限界 一通のコマンドメールの処理には、要求メールを受信する MTA である `sendmail`、コマンドを処理する `command.pl`、応答メールを整形する `compack.pl`、応答メールの送信を行なう MTA である `sendmail` の 4 つのプロセスが動作する。このため、数十通のコマンドメールを一度に送信された際に、サーバマシン上のプロセステーブルが溢れ、サーバマシンそのものを不安定にすることがあった。

6 今後の課題

今後の課題としては、以下のようなものがある。

- 応答メールの分割機能の実装
- 多数のコマンドメールが同時に到着した際に、コマンドメールをを待ち行列に入れ、プロセスが増え過ぎないように管理する機構の構築
- 特権モードの認証機能の強化
- 応答メッセージの日本語化

今後、各 ML の運用を続けながらこれらを解決する予定である。

参考文献

[1] 楯岡 孝道, “グループワーク支援システム ARGO,” 1995 年 調布祭 百万石, pp.1-4, MMA, 1995.

[2] 楯岡 孝道, “共同作業支援環境 ARGO の改良,” 1996 年 新入生歓迎シーズン 百万石, pp.3-7, MMA, 1996.

[3] “桜井智 ML 「智の会」の紹介,” 1995.
`http://www.wp.com/STREAM/mli.html`

[4] N. Borenstein, N. Freed, “MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies,” RFC1521, 1993.
`ftp://ftp.uec.ac.jp/pub/wwfs/RFC/rfc1521.txt`

[5] David H. Crocker, "STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES," RFC822, 1982.
`ftp://ftp.uec.ac.jp/pub/wwfs/RFC/rfc822.txt`

楽しいPGP

君島 秀征 E-mail:sanjigen@mma.club.uec.ac.jp

Abstract

PGP というファイルを暗号化 (encrypt)/復号化 (decrypt) する機能を持つツールがフリーで入手できます。

鍵輪 (key ring) や公開鍵 (public key)/秘密鍵 (secret key) といった不思議な言葉や "暗号" という言葉が持つ魅力と相まって、PGP は絶好の遊び道具です。

あなたが使っているマシンに PGP がインストールしてあることを前提に、PGP の簡単な使い方を紹介し、mail で他人に root の password を送る程度の知識を得ます。PGP の重要な機能である署名についてはここでは触れません。

1 PGP の概要

PGP による暗号化ファイルのやりとりに必要な、公開鍵/暗号鍵, user ID などについて簡単に説明します。

これから PGP を使おうと考えた A は、

1. 公開鍵/暗号鍵のペアを作らなければなりません (`pgp -kg`)。
2. 公開鍵を公開します[2]。秘密鍵は公開してはいけません。

をしなければなりません。

さて、これから A に秘密のメールを送りたい B はどうすれば良いかというと...

1. A の公開鍵を入手します。
2. A の公開鍵を、自分 (B) の公開鍵の鍵輪に追加します (`pgp -ka`)。
3. 入手した A の公開鍵が、本当に A の公開鍵かどうかを確かめます。普通は、公開鍵の指紋 (fingerprint) を電話などの確実な方法で A から入手し、自分の鍵輪の中にある A の公開鍵の指紋と比較します (`pgp -kvc`)。
4. **A の公開鍵で暗号化**します。
5. 暗号化したメールを A に送ります。

1.1 user ID について

pgp の操作にはよく user ID が用いられます。例えば、私 (Kimijima Hideyuki) が自分の公開鍵を`foo.asc` というファイルに取り出すには、`pgp -kxa kimijima foo` とします。

私の鍵の user ID は "`Kimijima Hideyuki <sanjigen@mma.club.uec.ac.jp>`" ですが、コマンドラインで指定する user ID は、目的の鍵の user ID の一部で OK です。全部を指定する必要はないですし、大文字小文字も区別されません。`pgp -kxa sanjigen foo` でも OK です。

2 ゲームの前に

まずは、ホームディレクトリで、`mkdir .pgp ; chmod 700 .pgp` し、それから`setenv PGPPATH ~/.pgp` しておいて下さい。

[2]MMA で PGP を使っている人は、みんな WWW page にも公開鍵を置いておいているようです。

3　楽しい PGP の公開鍵/秘密鍵の作り方

始めに、あなたの公開鍵/秘密鍵のペアを作らなければなりません。一度公開した鍵をおいそれと変更することはできませんから、この鍵はもしかすると一生使うことになるかも知れれません。心してとりかかりましょう。

`pgp -kg` で作ることができます。

1. 鍵の強さを決めます。鍵の強さは、この bit 数の累乗に比例するそうです。迷わずに 3 の 1024bit を選びましょう。あなたが軍人ではなくても、です[3]。

2. user ID を決めます。一般に、"`Kimijima Hideyuki <sanjigen@mma.club.uec.ac.jp>`" のように、本名とメールアドレスを user ID とします[4]。この形式を前提としているツールもあります。ここで天の邪鬼になっても良いことは無いので、素直にこの形式を守りましょう。
   一つの鍵のペアに複数の user ID を付けることもできますが、ここではまだ最初の一つしか付けられません。

3. パスフレーズ (pass phrase) を決めます。パスフレーズは、(お約束ですが) 覚えやすく人に推測されにくいものでなくてはなりません。パスフレーズは後から変更することもできますから、あまり悩まずに決めてしまいましょう。パスフレーズを忘れてしまうと、その鍵はもう使うことはできなくなってしまいます。実際、パスフレーズを忘れちゃった人[5]がいましたが、彼は新しく鍵を作り直して公開し直しました。しばらくは sanjigen にチクチクいじめられることでしょう。

4. 最後に、キーボードをランダムに叩き、乱数の種を作ります。十分な数だけ叩くと、鍵のペアが作成され、終了します。
   ただし、2 度目のパスフレーズの入力で間違った場合はこの過程は無いようです。

これで、あなたの公開鍵と秘密鍵が鍵輪に登録されました。

3.1　鍵のペアに複数の user ID を付けよう

メールアドレスがたくさんある人[6]は、自分の鍵にメールアドレスの数だけ user ID を付けたいかも知れません。`pgp -ke` で、今作った鍵のペアに複数の user ID を付けることができます。

1. `Do you want to add a new user ID (y/N)?` (新しい user ID を追加したいですか?) で`y` と答え、新しい user ID を追加し、

2. `Do you want to change your pass phrase (y/N)?` (パスフレーズを変更したいですか?) には`n` と答え、終了します。
   ここで`y` と答えれば、パスフレーズを変更することができます。

4　PGP の鍵の取り出し方

さて、PGP で遊ぶには[7]、あなたの公開鍵をみんなに配布しなければなりません。

`pgp -akx` [あなたの user ID] [ファイル名] で、あなたの鍵輪から公開鍵をファイルに取り出すことができます。このファイルをあなたの WWW page にでも置いておきましょう。

5　PGP の鍵の追加のしかた

友達からもらった公開鍵を使うには、あなたの鍵輪にその鍵を追加しなければなりません。`pgp -ka` [鍵のファイル名] で、鍵輪に追加できます。

[3]そうそう、bit は複数でも s は付かないのではなかったっけ。
[4]名前を "姓" "名" の順にしているのは、単に私のポリシーです。そうしなければならない理由があるわけではありません。
[5]名は秘するが、MMA の誰か。
[6]MMA の人はみんな当てはまるかな。
[7]もちろん、実用でも。

6　PGP の鍵の検証

友達 (A) からもらった公開鍵は、本当に A の公開鍵でしょうか。偽造されていたりはしないでしょうか。電子メールは、改変されているかも知れません。こういったとき、電話で公開鍵を伝えれば改変の心配は無いのですが... 面倒ですね。そこで、公開鍵を**もしかしたら改変されているかも知れない媒体** (電子メールとか WWW とか) でやりとりし、公開鍵の指紋だけを**絶対に安全な媒体** (直接会うとか電話とか) で確認する方法があります。指紋というのは 16Bytes の 16 進数の羅列です。

1. **もしかしたら改変されているかも知れない媒体**で A の公開鍵を手に入れる。
2. 自分の鍵輪に追加する。
3. **絶対に安全な媒体**で A の公開鍵の指紋を聞く。
   その A は、電話の向こうで A の公開鍵の指紋を`pgp -kvc` [A の user ID] で、表示しているでしょう。
4. 電話のこちら側でも`pgp -kvc` [A の user ID] で、A の公開鍵の指紋を表示する。
5. 聞いていた指紋と一致するか調べる。

指紋が一致していれば、偽造された鍵ではないと考えられます。

7　暗号化のしかた

あなたは、A という友達に秘密のファイルを送りたいとします。そのためには、ファイルを **A の公開鍵で暗号化**しなければなりません。

あなたが A の公開鍵を既に入手していて、鍵輪に追加してあるのなら、`pgp -ae` [ファイル名] [A の user ID] でファイルを暗号化することができます。

まだ A の公開鍵を入手していないのなら... まずは A の公開鍵を入手しましょう。

8　暗号の復号化のしかた

友達から PGP で暗号化されたファイルを受け取りました。おちろん、このファイルは、**あなたの公開鍵**[8]**で暗号化**されていることでしょう。

暗号化されたファイルを復号化するのは簡単です。`pgp` [暗号化されたファイル名] で復号化することができます。

9　楽しい PGP の注意です

9.1　メール改変作戦

ここに、A という人と B という人がいます。tobiishi というマシンの root の password が急に必要になった A は、それを知っている B に、次のような mail を送りました。

> A@某所です。
>
> ねえねえ、B クン。tobiishi の root の password を教えてよ。急に必要になったんだ。平文のままやりとりするのは危険だから、PGP で暗号化して送ってネ。ボクの公開鍵は...

さて、それを読んだ B は、このメールに添付されていた A の公開鍵を使って password を暗号化し A に送りました。公開鍵は誰に見られたって平気です。さらに、この鍵を使って暗号化した文を他人が盗み見ても、password は解りません。だから、彼女らは安全に password をやりとりできました...

本当でしょうか? ここに、C という人がいます。C は、A と B のメールのやり取りを盗み見ることができ、さらにはこっそり改変しちゃうことだってできる人です。

[8] あなたがみんなに配った鍵です。

1. A から B への mail を横取りし盗み読んだ C は、
2. 新たに公開鍵/暗号鍵のペアを作ります。
3. mail に添付されていた公開鍵を、新たに作った公開鍵に書き換えて B に送ります。
4. B は、mail に添付されている公開鍵を使って password を暗号化し、A に送り返しました。
5. その mail を再び横取りした C は、暗号化された password を復号化し、
6. それを本物の A の公開鍵を使って暗号化し、A に送りました。

これで C は、A にも B にもバレずに tobiishi の root の password を手に入れることができてしまいました。A と B が、PGP の公開鍵の指紋を電話で確認しあっていなければ[9]、の話ですけどね。

なぜ公開鍵が本人の物であるか確認しなければならないか、これで解りましたね。

9.2 パスフレーズ不正取得作戦

PGP を使っていると、パスフレーズを頻繁に入力します。X 端末の上で打ち込んだパスフレーズはネットワークの上ではツルツルです。ネットワークを監視していると、パスフレーズが見えちゃいます。

あなたが使っている PGP は、オリジナルの PGP ではないかも知れません。入力されたパスフレーズがどこかに記録されている... そんな仕掛けのある PGP かも知れませんよ。

10 徒然なるままに

自分の鍵の指紋は 012345...のように覚えやすいモノが良いなぁと思って、「ここは PGP の乱数生成ルーチンに手を加えてでも面白い鍵を作ってやるぜ!!」と考えました。しかし... それは不可能です。乱数生成ルーチンに手を加えるくらいで思い通りの指紋を持った鍵を作れるのなら、上に挙げたような悪いことは防ぎようがありませんね。鍵の指紋で偽造の有無を調べるという方法自体の意味が無くなってしまいます。

というわけで、こういう試みは不毛です。あきらめましょう。

参考文献

[1] pgp 2.6.3i ドキュメントファイル, ソースファイル

[2] UNIX MAGAZINE 1995 年 7–10 月号 “転ばぬ先のセキュリティ”

[9] 電話が使えるのなら、電話で password を伝えた方が早いよね (苦笑)。

Part II
ハードウェア編

PCBRIGEの作り方

関根康平 (eggman@mma.club.uec.ac.jp)

1 はじめに

PCBRIDGE とはピサ大学の luigi 氏[10] が作成された IBM-PC をブリッジにする DOS 上で走るプログラムです。バージョンが 3.0 になって FreeBSD の netboot コードを元に新たに書き直され、NE2000 がサポートされるようになりました。

市販のブリッジは、最近はあまり使われてないのか、高価でしかも 10baseT のポートを持っていません。PCBRIDE によって極めて安価にブリッジを作る事ができます。

2 ブリッジとは

難しく説明すると、OSI 参照モデルで言うデータリンク層を中継する装置です。ブリッジはただ中継するだけでなく以下のような機能を持っています。

- 蓄積転送

  あるポートから受け取った ethernet フレームを他のポートに転送します。そのとき、ブリッジは、受けっとったフレームを一旦メモリーに蓄え、出力ポートにコリジョンが発生していなければ、出力します。この機能によりブリッジで分割された片方のネットワーク上で起きたコリジョンをもう一方のネットワークに中継しません。またフレームを再生成するのでケーブル長の制限がなくなります。

- 学習

  ブリッジは受けっとった ethernet フレームの始点 MAC(Media Access Control) アドレス、終点 MAC アドレスを認識し、ホストがどのネットワーク上に在るかを自動的に学習します。その学習の結果を利用して、もし始点アドレス、終点アドレスが同じネットワーク上に在ればそのフレームは既に受信されているはずなので、ブリッジはそのフレームを破棄します。

- ループの検出

  ブリッジはループが起きるとフレームを増殖させる事になります。ループが起きたらブリッジは中継を停止します。

- IP 透過性

  ブリッジは IP からは認識できません。ブリッジを制御するのには、ブロードキャストや MAC アドレスを直接指定することによっておこないます。

3 作り方

用意するもの

- ネットワークインターフェイスカード (NE2000,3C509,3C503,WD8013)
- PC 一式 (最低限 386SX 以上の CPU,M/B,256K 以上のメモリー,FDD)
- bdg300.tgz [11]
- DOS [12]

[10] http://www.iet.unipi.it/ luigi
[11] http://www.iet.unipe.it/ luigi/FreeBSD/bdg300.tgz
[12] FreeDOS を今度使ってみようかな

ネットワークインターフェイスカードの IRQ, IO アドレスを、コンフリクトしないように設定する。(1 枚ずつ差して設定する) アーカイブを解き bdg303.com をシステムを転送したフロッピィーディスクにコピーし適当に autoexec.bat を書く。ケーブルを繋ぎ (カードからカードはクロスケーブルで接続する) 電源を入れと、コンフリクトなどがなければ PCBRIDGE が動作するはずだ。画面には、ネットワークの状況がモニターされる。

4 ベンチマーク

mola (P5-120 64M NE2000 BSD/OS2.1) kou (AMD 5x86 40 × 3 36M NE2000 LINUX2.0) pcbridge (386DX-40 4M NE2000,NE1000) 間を

mola – hub -100 メートルのケーブル– PCBRIDGE – hub – kou

の様につないで kou から flop へ ping,ftp を行った。

ベンチマークの結果		
	PCBRIDGE	NONE
ping -c 100	2.5ms	1.0ms
pinc -c 100 -s 1492	21.7ms	6.9ms
ftp (700k)	98kbytes/s	440kbytes/s

PCBRIDGE の NE1000 を NE2000 に変え ケーブルを短くしてみたら

ベンチマークの結果		
	PCBRIDGE	NONE
ping -c 100	2.0ms	1.0ms
pinc -c 100 -s 1492	15.5ms	6.9ms
ftp (700k)	260kbyete/s	440kbyte/s

と 2 倍程度速くなった

5 pcbridged

pcbrided とは PCBRIDGE をリモートでコントロールするためのプログラムです。今回は時間がなくて詳しくは書けないです。

6 その他の PC ベースの bridge

以下のの 2 つはファイヤーウォールの構築を目的としたフィルタリングブリッジです。

- karlbridge(http://www.karlbridge.com/)
  これは売り物ですが古いバージョンのシェアウェア版があります
- drawbridge(archie で探してね drawbridge-2.0.tar.gz)

7 まとめ

PCBRIDGE を使う事に、よって MMA 内の NFS 等のトラフィックを外部に流さなくなるが、ネットワークの性能の低下、信頼性の問題を考えると、通常時に使用するかは、まだ未定です。今後 club ドメインのホスト数が増えれば導入の必要性が増すと思われます。

参考文献

PCBRIDE の README, ソース
Internetworking 96.12 LAN 学事始め

汎用入出力インターフェイス SEIRA

楯岡 孝道 (tate@spa.is.uec.ac.jp)

Abstract

MMA において UNIX からさまざまな機器を制御したいという要求は多いが、UNIX から制御可能な汎用入出力インターフェイスは非常に限られているうえ、制御自体も難しいために実際の制御に成功している例は稀である。そこで、マイクロプロセッサを用いた汎用入出力インターフェイス SEIRA を設計した。SEIRA は RS-232C を用いてユーザの制御プログラムと通信を行いつつ実際の入出力を制御することにより、制御プログラムの記述を容易にする。

1 はじめに

過去 MMA においてさまざまな機器を制御したいという要求は数多く存在した。例えば、無停電電源からの割込みによる強制 shutdown の実現、部室内温度監視システムの構築、遠隔部室管理システムの操作などである。しかし、現在 MMA で主に利用されている UNIX 上から制御可能な入出力インターフェイスはプリンタポートと RS-232C 程度である。これらのインターフェイスを用いてもビット単位の制御を行なう制御は OS 毎に細かな違いがあり、実際の制御は難しい。

そこで、汎用入出力インターフェイスとして SEIRA (Simple Environment for Intelligent Remote Access) を設計した。SEIRA を用いることにより、RS-232C による通信を行なうだけで、ビット制御可能な入出力インターフェイスを得ることができる。

2 SEIRA システム概要

SEIRA はワンチップマイクロプロセッサを用いたワンボードコンピュータであり、通信用に RS-232C インターフェイスと、制御用に 64 ビット分の汎用入出力ポートを持つ。

SEIRA は ROM 上に基本管理プログラムを持ち、RS-232C を通じてコマンドを受けつける。コマンドによって特定ポートへの入出力を行うほか、RAM 領域に新たな制御プログラムを書込み、実行することにより、アプリケーションに適したインテリジェントな制御が可能になる。

SEIRA のハードウェアは容易に利用できることを考慮し、秋月電子のキットである「Super AKI-80」を利用する。これは Z80 上位互換 CPU である TMP Z84C015 を核に、2 チャンネルの RS-232C インターフェイス、64 ビット分のパラレルポートと、32K byte の RAM、8K ～ 32K byte の ROM などから構成されるワンボードコンピュータである。

これに基本管理プログラムの ROM を実装することで、RS-232C で制御可能な汎用インターフェイスとして利用する。

3 SEIRA 基本管理プログラム

SEIRA 基本管理プログラムは、以下のコマンドを持つ。

port read 指定したアドレスの I/O ポートの内容を読み出す

port write 指定したアドレスの I/O ポートに、指定したデータを出力する

port watch 指定したアドレスの I/O ポートの内容が変化するまで待つ

memory read 指定したアドレスのメモリの内容を読み出す

memory write 指定したアドレスのメモリに、指定したデータを書き込む

jump 指定したアドレスに CPU の制御を移す

これらの機能の他、SEIRA 上で動作するユーザプログラムの為に、RS-232C の入出力や文字列処理等を行なうサブルーチン群を用意する。

4　プログラム開発環境

SEIRA 基本管理プログラムを含む、SEIRA 上で動作するプログラムの開発は UNIX 上で動作する CP/M エミュレータを利用する。これは SEIRA が利用する TMP Z84C015 は Z80 上位互換であり、CP/M 上の豊富な開発環境で作成したプログラムが動作するためである。

5　SEIRA プロジェクト進展状況

SEIRA の実装を行なう SEIRA プロジェクトは現在、Frank Cringle 氏作成の CP/M エミュレータ yaze (Yet Another Z80 Emulator) を利用して基本管理プログラムの作成中である。

現在、コマンド処理部分は完成し、入出力ルーチンを作成している。SEIRA 基本管理プログラムは調布祭期間中の時間を利用して完成させる予定である。

残念ながら、現在 MMA には正常に動作する ROM ライタが無いため、ROM への書き込みは部外への依頼に頼ることになる。そのため、管理プログラムの作成には慎重を期する必要がある。

このような状況は開発環境としては最適なものではないため、今後 ROM ライタを作成し、SEIRA 基本管理プログラムの改良や、ユーザプログラムの ROM 化などが容易にできる環境の構築も必要である。

MOTOR DRIVE

中原 隆文

Abstract

ここでは、素人考えに乗っ取り、PC 上から簡単な機械的なシステムに作用させるためにステッピングモーターを制御することについて述べる。

1 スッテピングモーターとは

スッテピングモーターは電源に接続されただけでは駆動されない。VCC に電圧をかけた上で、複数ある極に順にパルスを供給してやることによって初めて動く。このようなしくみから、停止時に大きな静止トルクが得られ、回転する角度と速度を精確にコントロールすることができる。

2 仕様の決定

ここでは、ドアの鍵の開閉装置という、簡単な操作にステッピングモーターを利用する方法を例に、話を進めていく。PC[13]からのスッテピングモーターの操作を考えた場合、

- OS
- PC と外部デバイスの役割分担
- PC と外部とのインターフェイス

などで、幾つかの組み合わせが考えられる。

2.1 OS

本格的な制御システムを構築するなら ITORON, CHORUS, RT-MACH などのリアルタイム OS が必要とされるのであろうが、この事例では高々1 個のモーターを、それも比較的緩やかな時間制限の中で制御できれば十分なのでそこまでは必要ない。とりあえず、ここでは

- ユーザー認証の手段が標準で提供されている (オウセンティケーションを自分でプログラムしなくて良い)
- ハードウェアを簡単なプログラムでたたける (実効 uid が root にセットされているプログラムであれば、ioperm 関数でカーネルから特定の io port をたたく許可をもらえば、デバイスドライバーを介すことなく io port をいじれる)
- ネットワークが標準で利用できる (ネットワークごしに制御させたいこともあるかもしれない)

などの理由から、Linux をつかって話をすすめていく。

[13]PC といえば当然 AT 互換機である。

2.2 PC と外部デバイスの役割分担

次項の外部とのインターフェイスとも関係するのだが、PC 本体とスッテピングモーターを起動する外部デバイスとのどっちがどこまでやるかという処理の配分で決まる。考えられるパターンとしては、

1. すべてを PC 上でソフトウェア的におこない、外部デバイスはモーターへの電源の供給のみをおこなう
2. ISA, VME バス接続もしくはシリアルポート接続の専用のステッピングモータコントローラボードを使い、PC 本体は動作の指定のみを行い、処理の大部分はボードにまかせる
3. 上記二つの折衷
   PC はクロックの生成などモーターのコントールをある程度行うが、シリアルポートやパラレルポートを通じて接続した外部デバイスがクロックからのパルスの生成などよりモーターよりの処理を行う

の 3 つが考えられる。OS 同様さしてスペックはいらないので 1 か 3 が考えられるが、はじめにモーターのドライブ用にキットを用意していたので 3 の方法をもちいる。しかし、1 は外部デバイスの工作がかなり簡単[14]であるので、状況によっては 1 もおすすめである。

2.3 PC と外部とのインターフェイス

- 専用のスッテピングモーターコントロールボードの出力を使う
- シリアルポートを使う
- パラレルポートを使う

などが考えられるが、専用のボードは使用しないので、あとは、入出力電圧の使いやすさと出力線の多さからパラレルポートを使用する。

3 実装

モーターをコントロールするのに必要な手順は次の 5 つに分けられる。

1. 回転する速度と角度の決定
2. モーターにかける電圧と回転方向指定のスイッチ
3. モーターをドライブするための基準となるクロックの生成
4. そのクロックからモーターにあわせた矩形波を出力
5. その矩形波をモーターと駆動方式にあわせて増幅

ここで使用した秋月電子の “汎用・ステッピングモータ・ドライブ・キット 2 相励磁” は、ドライブに必要な次の処理の内、3, 4, 5 の機能があり、ただ回転させ続けるだけであればそのまま使用できる。しかし、ここでは回転速度も PC でコントロールしたいので、1, 3 を PC 側で、2 を 4 と 5 の回路の間に挟んだセレクタ TTL で行う。

3.1 外部デバイス

キットとセレクタ TTL を組み合わせた回路は次のとおり。

[14] ハード製作の項にある、“5. 矩形波をモーターと駆動方式にあわせて増幅” の回路のみでよい。

3.2 制御用プログラム

linux では OS のところでも書いたように、ioperm 関数でたたきたい I/O ポートの許可をとり、

```
void PortOutb(char value, u_short port)
{
    __asm__ ("outb %0,%1"
     ::"a" ((char) value),
     "d" ((u_short) port));
}
```

というような関数を用いてアクセスする。これでパラレルポートのコントロールレジスタ[15]を操作することによって、パラレルポートの制御線から外部デバイスをコントロールしたりクロックを供給したりすることができる。

ここではコントロールレジスタの下 1 ビット目をクロックの供給に、下 2 ビット目をセレクタ TTL のストローブ[16]に、下 3 ビット目をセレクタ TTL のセレクタに、使用している。

実際動作させるには、セレクタ TTL のストローブを low にし、セレクタ TTL のセレクタを回転させたい方向にあわせて set し、回転させたい分だけ回転させたいタイミングで high&low を切り替えてクロックを供給してやれば ok である。

ただ注意が必要なのは、クロックの供給が停止しても、どの極かの励磁信号が high になっていればモーターには電圧がかかっており静止トルクが生じているので、モーターの操作を終了するには、セレクタ TTL のストローブを high にするなどして、モーターに電圧がかからないようにしておく必要がある。

4 運用について

一般ユーザーもプログラムを実行することから、ioperm 関数には root 権限がいるので、プログラムは setuid され実効 uid が必要になる。

また、鍵の開閉と言うクリティカルな操作を実行させる場合、ネットワーク越しに侵入を許してしまったときの被害を考えると、セキュリティーに気を付ける必要があるだろう。

謝辞及び参考文献

スッテピングモーターを駆動するための回路の製作にあたっては、工学研究部の黒田さんに多大な協力をいただきました。それと、MMA 諸氏にはいろいろと資料を提供いただきました。

どうもありがとうございました。

参考文献

[1] 秋月電子 “汎用・ステッピングモータ・ドライブ・キット 2 相励磁” 説明書

[2] 東芝 “DynaBook テクニカルマニュアル [ハードウェア編]”

[3] Takashi MANABE (manabe@tut.ac.jp) kon の source

[15] 別にデータレジスタを使用しても良いのだが、読み書きの指定などがめんどうなのと、制御線が 4 つもあれば十分なので、ここではコントロールレジスタのみを使用する。外部センサーからデータなども受け取りたいなどというときは、適当にほかの制御線も使用してください。

[16] これを high にするとセレクタ TTL のすべての出力が low になるので、モーターの on/off に使える。

mola.mma.club.uec.ac.jp.private(172.21.139.43) の構築について

京極 大輔

もうだいぶ経つが、MMA にもとうとう Pentium マシンが導入された。しかし、導入にはさまざまな紆余曲折が存在した。そのあたりの経緯などについて述べたいと思う。

それまで MMA では beat、gont、さらに flop の三台のマシンが稼動し、beat 上には BSD/OS1.1 をインストールし各種サーバ (NFS、SMTP、HTTP、canna 等) を走らせ gont には NetBSD1.0 を入れて実験、X 端末に、flop では当時最新であった BSD/OS2.0 を入れてメーリングリストサーバをさせたり、フリーソフトウェアのコンパイルやネットワークの検証をしてみたりといったように利用していた。

その後、部員の物納品やバイト先からのジャンク流れ品の PC パーツが MMA に集まりはじめ、メモリを除いて PC 一台分の部品が集まってしまった。そこで kou という完全実験用の遊びマシンを組み、FreeBSD が導入された。

順調にマシンが増え、そろそろ 486DX2-66 だった beat の負荷が重くなってきたと感じられるようになった頃であった、例の盗難事件が発生したのは。

入試のために休講になったその翌日、朝部室に行くとなんと flop がきれいさっぱり消えてなくなっていたのである。ただし flop に使っていた 5” フルハイトの HDD はわざわざケーブルが外されて残されていた。まぁ代わりに gont の HDD が盗まれていたのであるが、当時 gont は OS のバージョンアップに伴って停止していたためにしばらく盗難に気づかなかった。ともあれ、犯人がデカくてトロい 5” フルハイトを残したので gont にその HDD を繋げて仮復旧したのであるが、結局現在も flop は元 gont のケースに収まったままである。また、kou もメモリを盗まれてしまったために起動不能となっていた。

この事件で実質使えるマシンが beat と flop の 2 つだけになってしまって部室にいってもコンソールにありつけなくなってきた。しかも不幸は重なるもので beat の HDD の調子が怪しくなってきてしまった。起動時の fsck が通らなくなってしまい、しかも何度 fsck してもその度に新しく不良ブロックが見つかるのである。取り敢えず/etc と/usr/local、/home をたまたま手に入った 540MB の HDD に移動したもののシステムは相変わらず不調であった。そして部員たちの不満が新サーバ構築計画を発動させたのである。

新サーバはともかく速度より安定重視を念頭に設計された。

CPU は Pentium-120MHz、マザーに Intel の Endever、HP の HDD、ADAPTEC の SCSI カード、MATROX の 2 世代前のビデオカードとできるだけメジャーでちょっと古くても安定した (と思われる) ものを予算と相談の上で選んでいった。OS は beat からの各種サーバの移動を考えて BSD/OS2.1 を入れることにした。また、コンソールは重たい X クライアントソフトをたくさん動かされそうな気がしたので VGA しか出さないこととなった。もっとも当初は xdm すら動かさずにサーバに徹する予定ではあったのであるが...

さて、ここに MMA の当時残っていた予算をほぼ全てつぎ込んだサーバマシンが完成した。ちなみに “mola” の由来についてだが深く語るつもりはない。キーワードは「前世、海、東京テレメッセージのポケベル」である。どうしてもわからないという人は、世の中には知らない方が幸せなことが多いということできれいさっぱり忘れてしまうことが最善であろう:)

こうして mola が出来上がった。現在、これで NFS と SMTP、DNS のサーバサービスを行っており、HTTP もいずれ移動する予定である。

ところで先日、mola の nfs がちょくちょく落ちるようになってしまった。原因がいまいちよくわからなかったこともあり、HDD を換えるという話が出たのであるが、バッファを増やした所なんとかうまく動いているようである。しかし、それに伴い内部に入れられたままの HDD がやたら発熱し、またもともと入っていた HDD も発熱するタイプでもあった事から、mola の前面にファンが取り付けられる事となった。いずれ HDD を一台にまとめる予定のために今の所仮設のファンが付いているが、事がすみ次第消えることになっている。

またいままで ISA バスで Ether カードを使っていたためにパフォーマンスが落ちていたとも考えられるためにこのほど PCI の Ether カード、DEC の DE500 を導入する運びとなった。これでシステムが安定してくれると良いのであるが。

いまのところ MMA で Pentuim のマシンは mola 1 台だけであるが、いずれ徐々に他のマシンも速いマシンにシフトしていくことであろう。その時に mola の構築課程がそれらを立ち上げる時のなにかの役に立つなら幸いと思う。

以上

Part III
プログラミング編

とりあえずネットワークプログラミング
超簡易 chat サーバープログラミング

uirou@mma.club.uec.ac.jp

1 はじめに

ネットワークを利用したサーバープログラムは、実は思ったより簡単に作成する事ができます。ということで、とりあえず複数のクライアントからの入力を、全ての接続中のクライアントに返すという、いわば chat の簡易版みたいなサーバーを作ってみましょう。(今回作るのはサーバーだけです。クライアントには telnet を使用します)

2 まずは socket(2)、bind(2)、listen(2)

ネットワークの通信をするためには、まずは通信を受け取る (または送り出す) ための入口 (または出口) を作ってやらなければいけません。それを作るためのシステムコールが socket(2) です。socket の定義は、こんな感じで、

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol)
```

となっています。`socket` は、作成されたファイルディスクリプタを返します。

第一引数の`domain` には、`AF_UNIX` か、`AF_INET`、その他が入ります。それぞれの意味は、次のようになっています。

`AF_UNIX` UNIX ドメインの通信をするためのソケットを作ります。要するにおんなじホスト間でのプロセス通信に用いられます。Xserver などは、これを使って、`/tmp/.X11-unix/X0` なんていうファイルを作っています。

`AF_INET` INET ドメインの通信をするためのソケットを作ります。要するに、ネットワークを利用してプロセス間通信をしたい場合は、こちらを指定します。

今回のサーバーは、ネットワーク越しにお話をしたいので、`AF_INET` を使うことになります。

第二引数の`type` には、通常、“`SOCK_STREAM`” と、“`SOCK_DGRAM`” の二種類の内のどちらかが入ります。二種類の違いは以下の通りです。

`SOCK_STREAM` データが送り出された順番の通りに到着する事が保証されています。そのため、接続が確保されるまでは、データを送受信することはできません。

`SOCK_DGRAM` データはパケット単位で送出されます。パケットが送り出された順番に到着する保証はありません。それどころか、パケットがきちんと届くという保証すらないのです。そのかわり、接続を確保する必要がありません。

今回作成するサーバーは、簡易型 chat なのですが、一応、データの順番は送り出された順に到着してくれる事を望んで、`SOCK_STERAM` を使う事にします。

最後の第三引数には、プロトコルが入るのですが、ここには、`0` を指定します。というのも、`0` を指定する事によって、システムが適当なプロトコルを勝手に割り当ててくれるからです。

ということで、`socket` を呼び出すところを書いてみると、

```
int sock;
if( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ){
perror("socket");
exit(1);
}
```

という感じになります。

ソケットを作ったら、次に bind(2) を使ってソケットにポート番号を割り当てます。

bind(2) は、以下のように定義されています。

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int s, struct sockaddr *name, int namelen);
```

第一引数のs は、対象となるソケットのファイルディスクリプタを指定します。普通、socket(2) で作成したファイルディスクリプタをそのまま渡すことになります。

第二引数には、バインドするデータの指定をするためのsockaddr 型変数を指定します。sockaddr 型変数は、ソケットを作成した時に第一引数で指定した、ドメインによって与えるsockaddr 型変数が変わってきます。どのように変わるかといえば、次のように変わります。

sockaddr_un AF_UNIX を指定した時に使用されます。<sys/un.h>で定義されています。

sockaddr_in AF_INET を指定した時に使用されます。<netinet/in.h>で定義されています。

今回のサーバーでは、AF_INET を指定したので、sockaddr_in を使用することになります。さて、sockaddr 型変数で、何を指定しなければいけないかと言えば、sockaddr_in 型変数で、最低限設定しなければいけないのは、sin_family, sin_port,
sin_addr.s_addr の三つです。

プログラムでは、以下のような感じになります。

```
struct sockaddr_in sin;

sin.sin_family = AF_INET; /* AF_INET を指定 */
sin.sin_port = htons(3676); /* ポート番号をネットワークバイトオーダで指定 */
sin.sin_addr.s_addr = INADDR_ANY; /* 接続を受け付けるネットワークアドレスを ANY に指定 */
if( bind(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0){
perror("bind");
exit(1);
}
```

ここで、けっこうよく陥ってしまう点が、指定したポート No が既に使われている、というエラー (Address already in use ってやつ) です。自分は間違っていないのに... と、悩んでしまいがちです。注意しましょう (この例では、3456 を使用してみました)。

次に、listen(2) を使用して、bind されたソケットをクライアントからの要求を受けつけるように指定します。listen は、以下のように定義されています。

```
#include <sys/socket.h>
int listen(int s, int backlog)
```

第一引数のs は、指定されるソケットのファイルディスクリプタを指定します。まぁ、bind されたファイルディスクリプタをそのまま渡してしまいましょう。

第二引数のbacklog は、一度に要求を受け取るクライアントの数を指定します。たいていは 5 以下の値に設定されているようです。[17]

プログラムにすると、以下のような感じになります。

[17] まぁ、たいして重要でもないので、5 にでもしておけばいいのでしょう。

```
if(listen(sock, 5) < 0){
perror("listen");
exit(1);
}
```

では、これらの一連の動作を、一つの関数にしてみましょう。[18]次のような感じでいいはずです。

```
int initsocket(int port){
    int sock;
    struct sockaddr_in sin;

    if( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ){
        perror("socket");
        exit(1);
    }
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);
    sin.sin_addr.s_addr = INADDR_ANY;
    if( bind(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0){
        perror("bind");
        exit(1);
    }
    if(listen(sock, 5) < 0){
        perror("listen");
        exit(1);
    }
    return(sock);
}
```

この関数には、オープンするソケットのポートを引数として与えると、オープンされたソケットのファイルディスクリプタを返してくれます。エラーがあった場合は、プログラム事態を終了するようになっています。要するに、エラーで返る事は無いというわけです。まぁ、変な仕様ですが、サーバーの実験用であるのだから、あまり問題にはならないでしょう。

これで、インターネットドメインでソケットを作り、クライアントを受け入れる準備ができました。次の章では、複数のクライアントを効率良くさばくための select システムコールの使用法を説明します。

3 select しなきゃ、とまっちゃう

前の章では、`socket` でソケットを作り、`bind` でポート番号を割り当て、`listen` でクライアントからの入力を受け付けました。

この章では、accept(2) を使って、クライアントからの接続を受けたり、select(2) を使って、クライアントからの入力を上手にさばく方法を説明します。

まずは、accept(2) システムコールから説明しましょう。

accept システムコールは、クライアントからの接続要求を受け付けるためのシステムコールです。実際の動作は、すでに listen されているソケットに対して、接続を受け付け、接続されたソケットに新しいファイルディスクリプタをつけて、その番号を返す、ということをします。以下のように定義されています。

```
#include <sys/types.h>
```

[18] まぁ、決まりきった操作なので、一つの関数にしてしまった方が良いでしょう

```
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, int *addlen)
```

第一引数のs は、接続を受け取るソケットのファイルディスクリプタです。第二、第三引数は、接続してきたクライアントの情報を得るためのsockaddr 系変数へのポインタと、そのサイズです。が、今回はとりあえずクライアントの情報はいらないので、両方とも NULL を指定しておきます。

実は、accept は、クライアントからの接続がないと、ブロックしてしまい、処理が止まってしまいます。これを回避するためには、select(2) を使います。

select システムコールを使えば、ブロックする事は無く、複数のファイルディスクリプタに対して、入力, 出力, 例外条件が発生したかどうかを調べる事ができます。

select システムコールは、以下のように定義されています。

```
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exeptfds,struct timeval *timeout);
```

第一引数のnfds は、チェックするファイルディスクリプタの数を指定します。第二、三、四引数は、それぞれ、読み出し, 書き込み, 例外的条件をチェックするためのファイルディスクリプタ集合へのポインタを指定します。最後の第五引数は、select でチェックする時に待つ時間を指定します。もし、ここに NULL ポインタを与えると、第二〜第四引数で指定したどれかが成立するまで待ち続けます。

今回のサーバーでは、クライアントから読み出すものをチェックしさえすれば良いので、(書き出しの方は、チェックしないでもいいでしょう。たぶん :) writefds, exptfds, timeout には、すべて NULL ポインタを与える事にします。また、第一引数の値は、最大値がFD_SETSIZE 個と決まっているので、それを使わせてもらう事にします。

さて、readfds の値は、どのように設定すれば良いのでしょうか。実は、select(2) のマニュアルページを読むと、FD_SET, FD_CLR, FD_ISSET, FD_ZERO というマクロの事について書いてあるのがわかります。こいつの仕様は、以下のようになっています。

FD_SET(fd, &fdset) fd で指定されたファイルディスクリプタを&fdset で指定されたfd_set 構造体に設定します。

FD_CLR(fd, &fdset) fd で指定されたファイルディスクリプタを&fdset で指定されたfd_set 構造体から削除します。

FD_ISSET(fd, &fdset) fd で指定されたファイルディスクリプタが&fdset で指定されたfd_set 構造体で設定されているかを調べます。

FD_ZERO(&fdset) 指定されたfd_set 構造体をクリアします。

select は、返り値に、準備のできたファイルディスクリプタの総数を返します (エラーの場合は、-1 が返ります)。どのファイルディスクリプタが準備完了かというのは、指定したfd_set 構造体の中に入って帰ります。要するに、指定した構造体の中身を見て、判断し、結果も同じ構造体に入って帰って来るという事になります。そのため、fd_set 構造体事態のデータを、一度とっておかねばなりません。

さて、select して、accept すれば、クライアントとファイルディスクリプタで接続できるので、後は fdopen(3) を使って、fprintf などが使用できるようになりますね。

が、一応SOCK_STREAM で定義された socket 用のread, write, close システムコールが用意されているので、それを説明しておきます。

recv(2) は、socket 用の read(2) です。read と違うのは、最後に特別なオプションを指定できる点です。これらのオプションは、以下の三つです。

MSG_PEEK これを指定すると、そのソケットから、データを「のぞき見る」ことが出来ます。実際、のぞき見たデータは、もう一度読み出す事が出来ます。

MSG_OOB これを指定すると、そのソケットに送られた、例外データを受け取る事が出来ます。select の第 4 引数で引っかかった場合は、これで送られて来るのですね。

MSG_WAITALL これを指定すると、データが全て送られて来るまでブロックします。要するに、データを全て受け取ろうと言うのですね。

send(2) も、socket 用の write(2) です。write と違うのは、これまた、最後に特別なオプションを指定できる点です。このオプションは以下の一つです。

MSG_OOB これを指定すると、そのソケットに例外データを書き出すことが出来ます。このデータは、他のデータを飛び越えて、読み出す事が出来ます。非常事態等の時に使用するわけですね。select の第四引数に引っかかるデータを送り出せるわけです。

shutdown(2) は、socket 用の close(2) です。こいつも、最後に特別なオプションが指定できます。これらは、値で制御されます。[19]また、shutdown を使用すると、現在受け取っていないデータは捨てられます。もしも、close システムコールを使用すると、オペレーティングシステムが輸送中のデータを届けようとしている間はブロックされてしまいます。

0 を指定すると、ソケットは、読み出しにだけクローズされます。

1 を指定すると、ソケットは、書き込みにだけクローズされます。

2 を指定すると、ソケットは、読みだし、書き込み、両方に対してクローズされます。事実上、接続は解除されます。

ということで、大体こんな感じで使えば良いという事になります。

```
int sock;
int client_fd[MAX_CLIENT];
int fdnum,clients;
fd_set read_ready,read_orig;
char data[MAX_DATA];

int i,j,n;

/* これは、さっき作った関数ね */
sock = initsocket( 3456 );

/* とりあえず、初期化する */
bzero( &read_orig, sizeof(read_orig) );
FD_ZERO( &read_orig );
FD_SET( sock, &read_orig );

fdnum = 0;

while(1){
    /* 内容が無くなっちゃうので、オリジナルをコピーして select に渡す */
    bcopy( &read_orig, &read_ready, sizeof( read_ready ) );
    if( select( FD_SETSIZE, &read_ready, NULL, NULL, NULL ) > 0 ){

        if( FD_ISSET(sock, &read_ready) ){
            /* ソケットから読み込めるという事は、そのまま接続要求という事 */
            if(fdnum >= MAX_CLIENT){
            j = accept( sock, NULL, NULL );
```

[19]なんで define してくれないのかなぁ

```
            shutdown( j, 2 );
        }else{
            client_fd[fdnum] = accept( sock, NULL, NULL );
            /* オリジナルに登録 */
            FD_SET( client_fd[fdnum], &read_orig );
            fdnum++;
        }
    }

    for(i = 0; i < fdnum; i++){
        if(FD_ISSET(client_fd[i], &read_ready)){
            if( (n = recv(client_fd[i], data, sizeof(data), NULL)) >= 0){

                if(data[0] == 4 || n == 0){
                    /* もしも、クライアントが接続を切ったら、ソケットをクローズする */
                    shutdown(client_fd[i], 2);
                    /* オリジナルから登録を削除 */
                    FD_CLR(client_fd[i], &read_orig);
                    /* クライアントのリストからも、削除 */
                    client_fd[i] = client_fd[fdnum - 1];
                    fdnum--;
                }else{

                    /* ここで、書き込まれたデータを現在つながれているクライアントに書き出す */
                    for(j = 0; j < fdnum; j++){
                    send(client_fd[j], data, n, NULL);
                }
            }
  }
}
    }
    }
}
```

これで、一応サーバーとしての機能はできました。

次の章では、サーバーとしてプログラムが上がった時のエチケット (のようなもの) を説明します。

4 デーモンのを作る時には、これらに注意しようね

前の章では、accept(2) システムコールを使ってクライアントからの接続を受け付けたり、select(2) システムコールを使ってクライアントからの入力を上手にさばく事をおぼえました。

この章では、サーバープログラムがしておいた方が良い事を説明します。実は、今回プログラムしてきたサーバープログラムが、広く利用される事はありえ無いのですが、まぁ、一応、もしも広く利用して欲しいサーバープログラムを作るような事があったら、このような事をするのだよ。ということを説明したいと思います。

4.1 いらないファイルディスクリプタはクローズしよう

まず、サーバープログラムが起動された後、標準入力や、標準出力を使用しないのであれば、これらはクローズしてしまいましょう。もちろん、-debug が指定された時など、なにかの出力が使いたかったら、クローズしてはいけません。また、もしかしたら、親プロセスが他のファイルディスクリプタをオープンしているかも知れないので、全てクローズしておいた方が良いでしょう。それには、以下のようにする事で全てクローズする事が出来ます。

```
/* sysconf で、オープンできる最大のファイルディスクリプタの数を得る */
fd = sysconf(_SC_OPEN_MAX);
for (; fd >= 0; fd--)
    close(fd);
errno = 0; /* エラーをクリアしておく */
```

4.2 デーモンの親は init がいいよね

サーバープログラムは、大抵ずーっと動いている事が多いので、バックグラウンド実行されますよね。ですが、いちいち&をつけて起動させたりするよりも、プログラムが勝手にバックグラウンドになってくれるとかっこいいですよね。これは、実は結構簡単に実現できます。単純に fork(2) して、親プロセスが先に終ってしまえば良いのです。シェルは、バックグラウンドで実行しなかった場合、そのプロセス、すなわち、fork した後の親プロセスの終了を待ちます。そこで、fork して、親プロセスが終了する事によって、シェルは次の動作に移る事が出来ます。プログラムにするとこんな感じです。

```
if(fork() != 0)
    exit(0);
```

まぁ、もしも fork がエラーを返した時にも、ただ実行が終了してしまうので、できれば、

```
if( (pid = fork()) < 0){
    perror("fork");
    exit(1);
}else if(pid != 0){
    exit(0);
}
```

くらいには、した方が良いでしょう。

4.3 umask をきちんと設定しよう

サーバープログラムが作成するファイルなどは、それを実行したユーザーの umask が使用されます。すなわち、どんな umask が指定されているかわからないわけです。これは、ちょっと気持悪いですよね。ということで、一応能動的に指定してしまいましょう。

例えば、こんな感じですね。

```
umask(\077);
```

4.4 カレントディレクトリを変えようよ

普通、プログラムが起動されると、そのプログラムのカレントディレクトリは、実行されたディレクトリになります。これがもし、何らかの方法でマウントされたディレクトリだったとしたら、アンマウントできなくなってしまいます。(Device busy と言われてしまう) ということで、アンマウントの必要無いルートディレクトリにでも変更してしまいましょう。

例えば、こんな感じです。

```
chdir("/");
```

4.5 シグナルにも気を配ろうよ

よく、デーモンに HUP シグナルを送りますよね。そうすると、デーモンは、設定ファイルをもう一度読み直してくれたりしますよね。でも、これは、signal(3) を用いて指定しないといけませんよね。ということで、お行儀のよいデーモンプログラムを組むのなら、signal(3) を用いて、SIGTERM, SIGHUP, SIGINT, SIGCHLD, SIGTTOU, SIGTTIN, SIGTTSTP のシグナルは処理してあげた方が良いでしょう。

4.6 制御端末を切り離そうよ

デーモンプロセスが起動された場合、大抵は、/dev/tty を制御端末に持っています。実は、この制御端末の状態によってシグナルが送られる可能性もあったりするし、制御端末はデーモンにとって無ければならない物でもないので、これを切り離してしまいましょう。実際の操作は、以下のように ioctl(2) を使用します。

```
if( fd = open("/dev/tty", O_RDWR) >= 0){
    ioctl(fd , TIOCNOTTY, (char *)NULL);
    close(fd);
}
```

また、もしもあなたの UNIX システムがセッションという概念を導入していたら、新しくプロセスグループを作成し、そのセッションリーダーになることでも、制御端末を切り離す事ができます。これは、以下のようにすれば OK です。

```
setsid();
```

こっちの方がちょっと簡単ですね。

4.7 ログは、どうすんのよ

さて、これまで説明してきて、エラーの時には、どこにエラーを吐けばいいのよ。と考えた人もいるでしょう。その疑問はもっともです。実は、そんな人のために、syslog というシステムがあったりします。まぁ、/dev/console をオープンしたり、(制御端末にしたくないのであれば、open の第二引数に O_NOCTTY オプションを設定しましょうね) 特定の logfile に書き出しても良いのですが、とりあえずここでは、かっこよさそうな syslog システムを説明する事にします。

syslog(3) は、syslog.h と、varags.h で、以下のように定義されています。

```
void syslog(int priority, const char *, ...);
```

priority には、以下のログの重要度と、どのようなクライアントからのメッセージかという値とを論理和で結んで指定します。

ログの重要度

LOG_EMERG パニック状態

LOG_ALERT 即座に直さねばならない状態

LOG_CRIT 危険な状態

LOG_ERR エラー

LOG_WARNING ウォーニング・メッセージ

LOG_NOTICE 処理した方が良い状態

LOG_INFO インフォメーション

LOG_DEBUG デバグ用のメッセージ

クライアントの種類

LOG_AUTH ユーザー認証を行うプログラム (login, su, getty など)

LOG_AUTHPRIV LOG_AUTH と一緒だが、限られたユーザに情報を公開したい場合

LOG_CRON cron 等のデーモン

LOG_DAEMON ここに上げられていない普通のデーモン

LOG_FTP FTP 関連のプログラム (ftpd など)

LOG_KERN カーネル (通常プロセスは指定できない)

LOG_LPR プリンタ関連 (lpd など)

LOG_MAIL メール関連 (sendmail など)

LOG_NEWS ニュース関連

LOG_SYSLOG syslogd 用

LOG_USER ユーザープロセス用

LOG_UUCP UUCP 関連

LOG_LOCAL[0-9] 任意のプログラムに割り当てられる

また、メッセージの後ろは、printf 文と同じように書く事が出来ます。それから、“%m” という物が追加されています。これには、現在のエラーメッセージが入る事になります。(perror で出て来るのと同じです)

ということで、例えば、以下のように使用すれば良い事になります。

```
syslog(LOG_ERR | LOG_USER, "%m");
```

5 ふぅ。やっと終った。

さて、これでだいたいサーバープログラムを組む事が出来るようになったと思います。まぁ、4 章のような事は、最初のうちは、あんまり気にしないで、がんがんプログラムを書いていってしまってもいいでしょう。(多分)

とにかく、ここで説明したのは、サーバープログラムの基礎です。結局、サーバープログラムに何をさせるのかとか、プロトコルの事とかを考えるのはあなたです。頑張ってくださいね。

それと、今回作成したサーバープログラムは、

http://www.uec.ac.jp:8081/club/mma/~takkun/hyakuman/9611/server.c

から、get できます。よろしかったら、ご利用ください。

参考文献

参考文献

[1] 各種 man ページ

[2] NUTSHELL HUNDBOOKS UNIX C プログラミング

[3] UNiX MAGAZINE 1995/10〜1996/05 UNIX 流プログラミング

ということで、実は上に書いてある物を全部読むと、これよりもずっと知識がついたりします。ちょっと分からない事があったりしたら、読んでみてくださいね。

L I L I T H

～nlith と某人型汎用決戦兵器的精神攻撃動画へのオマージュ～

Version 0.02

So-Miya(miyaso-m@jed.uec.ac.jp) in MMA

1 書式

```
lilith {-[lwds]<num>|-t[0bo12345-]}
```

2 前置

旧ソ連で作られ、各マシン上のゲームとして人気を博したことがあったような気がする「TETRIS」のクローンの「nlith」にちょっと似てる気がするゲームです。

「TETRIS」では 4 つのタイルからなるブロックをうまく積み重ねていくものでしたが、「LILITH」では「nlith」と同じくタイル数は 1～5 です。

ルールは簡単で、落下してくるブロックをうまくコントロールし、なるべく隙間なく敷き詰めていきます。横一列にブロックが詰まっている列は、消去されます。列を消しきれずに敷き詰めたブロックと出現したブロックが重なってしまうと核融合反応がおきて祝福された死を迎える (俗に言う “補間される”) 事ができます。

なお、一度に多く列を消去すればするほど高得点が稼げるようになっております。

3 操作

以下のキーによって操作します。

4, h	...	ブロックを左に動かす
6, l	...	ブロックを右に動かす
5, k, z	...	ブロックを 90 度左に回転させる
8, m, x	...	ブロックを 90 度右に回転させる
2, j	...	ブロックを一列分落下させる。得点が 1 点ずつ入ります
SPACE	...	ブロックを何かにぶつかるまで落下させる。落した分だけ得点が入ります。なお、接地した後しばらくブロックを移動できます。
^C	...	中止する
q	...	終了する

4 神託表示機能

指定されたパスに、「lilith.msg」が存在すれば、ときおりウィンドウが開いてあなたに神託を与えることでしょう。しかし、実際には逆に脱力してしまう神託を授かる事もままあるようです。

5　オプション

$-l <$ 数値 $>$	ブロックを落とす箱の高さを設定します。$<$ 数値 $>$に 5〜40 指定できます。デフォルトは 20 です。ただし、画面を越える値は設定できません。
$-w <$ 数値 $>$	ブロックを落とす箱の幅を設定します。$<$ 数値 $>$に 5〜40 指定できます。デフォルトは 20 です。ただし、画面を越える値は設定できません。
$-d <$ 数値 $>$	神託を授かる頻度を設定します。1/$<$ 数値 $>$の確率でブロックが落下してくるごとに神託を授かります。デフォルトは 40 です。
$-m <$ 数値 $>$	1 ライン落下するまでにブロックが動かせる最大回数を設定します。デフォルトは 100 です。実際にはキー入力の取りこぼし等で同じ数だけ動かせるとは限りません。
$-t <$ 設定 $>$	ブロックの種類を設定します。$<$ 設定 $>$には $0, b, o, 1, 2, 3, 4, 5$ を複数指定できます。それぞれの文字の後ろに '$-$' を置く事によって逆の意味にする事ができます。

$<$ 設定 $>$	説明	'$-$' 時説明
0	ブロック未設定の状態	全て設定した状態
b	基本ブロックを追加	基本ブロックを削除
o	斜めブロックを追加	斜めブロックを削除
s	分離ブロックを追加	分離ブロックを削除
1	1 タイルのセット追加	1 タイルのセット削除
2	2 タイルのセット追加	2 タイルのセット削除
3	3 タイルのセット追加	3 タイルのセット削除
4	4 タイルのセット追加	4 タイルのセット削除
5	5 タイルのセット追加	5 タイルのセット削除

6　注意事項

本製品は食べられません。お子様の手の届かない場所に補間`^H^H`保管してください。

誤って目に入れた場合はすぐに水で洗い流した後、お近くの病院で検査を受けてください。

このバージョンはベータ版ですので配布は禁止します。(まぁ、こんなゲーム配布する奴はいないでしょうが)

このプログラムまたはその一部を使用したために生じた事故等について、原作者は一切の責任を問われないものとします。つまり、マシンがディラックの海に沈もうが、ゲームをプレイしたものが精神汚染されようが、原作者は一切関知しません。

7　プログラムについて

このプログラムは termcap を使用しております。

このプログラムは signal 命令と usleep 命令を使用しております。

このプログラムは curses を使用しておりません。

このプログラムは、いちおう kterm 上での動作を前提として作られています。しかし、漢字が表示できて、TERM が使えて、漢字が表示できる UN*X ならば動作するかも知れません。

このプログラムは MSDOS 上でもコンパイルできるように作成しましたが、MSDOS 上では termcap は使用しておりません。機種毎にコンパイルしてください。ただし、動作保証はまったくありません。特に、usleep 関数が機種によって実行速度が変わってしまってまったく意味がありません。

7.1　termcap の使い方講座

このプログラムを作成するにあたって、letters の termcap の部分を大いに参考にさせてもらいました。

まず、termcap を使用するには、"termio.h" をインクルードし、コンパイルするときに`-ltermcap`でライブラリを付加しなければならない事を覚えておいてください。

プログラムの中では、次のようにします。

```
char *term_name;

init_term(){
  char tbuf[1024];

  if((term_name = (char *)getenv("TERM")) == NULL) {
    /* エラー処理 */
  }
  if(tgetent(tbuf, term_name) <= 0) {
    /* エラー処理 */
    exit(1);
  }
  .
  .
  .
```

上のプログラムではまず、環境変数 TERM から端末名を取得し、次にそれにたいするエントリを tbuf に格納しています。tgetent の第一引数で指定したバッファが以後の termcap 関数で使用されます。

これを実行した後、`tgetnum(char *id)` や`tgetstr(char *id, char **area)` 等を呼び出します。

例えば、画面の幅と高さを得るには

```
  Lines = tgetnum("li");
  Columns = tgetnum("co");
```

とします。この時の”li” や”co” の解説は`man 5 termcap` すると載っています。

tgetnum は数値を返してくれます。

次に、画面のカーソル位置を指定する方法です。

まず、

```
  *cursor_address = '\0';
  tgetstr("cm", &cursor_address);
```

として、カーソルアドレスを取得します。なお、`cursor_address` はグローバルな文字列変数です。

こうしたあと、

```
#define gotoxy(c, l)    putp(tgoto(cursor_address, c, l))
```

とでもしておけばカーソルの移動が出来るようになります。

くわしくは、各自いろいろなソースを見て研究してください。

grobot で遊ぼう

uirou@mma.club.uec.ac.jp

1 grobotって、なぁに？

grobot とは、uncover さんが作ったゲームで、一言で言うと自分で作ったプログラムを他人の作ったプログラムと戦わせようというゲームです。

grobot での行動は、索敵, 移動, 攻撃の 3 つに分けられます。あなたは、プログラムでこの 3 つのコマンドを駆使して敵を全て破壊すれば良いのです。

grobot1.3 では、C 言語でのヘッダファイルとライブラリと、perl5 による perl 用ライブラリが用意されています。

urobot とかいう名前もちらほら聞かれますが、あれは無視して、grobot13 ということでお話を進めます。

2 作ってみよう

まずは、`mkgrobot.*`に載っている、サンプルプログラムを打ち込んで動かしてみましょう。とりあえずサンプルプログラムはこうでした。

```
 1 #include <stdio.h>
 2 #include "../grobot.h"
 3 #include "sample.bitmap"
 4
 5 #define RADARMAX 8
 6 SCAN radar[RADARMAX];
 7
 8
 9 main( argc, argv )
10  int argc;
```

```
11  char **argv;
12 {
13
14 int id;
15 int x, y;
16 int misc;
17 int n;
18
19 id = gr_init( "sample", argc>1?argv[1]:NULL, '*', sample_bits );
20 while ( 1 ) {
21 n = gr_scan(radar,RADARMAX);
22 if ( n > 1 )
23 gr_missile(radar[1].x,radar[1].y,4);
24 }
25 }
```

もちろん、#include などは、プログラムのおいてある場所によって書き換えて下さいね。

また、多分あなたの環境には、sample.bitmap というファイルは無いと思います。その時には、bitmap(1) コマンドを使って、ビットマップファイルを作るか、少し後で書いてある、nerv.bitmap をsample.bitmap として打ち込んで、sample.c と同じディレクトリにセーブして下さい。

では、このプログラムをコンパイルしてみましょう。だいたいは、以下のようなコマンド入力でコンパイルは完了です。

```
% gcc sample.c -o sample -lgr
```

これでエラーが出る場合は、プログラムの打ち込みミスでなければ、grobot.h と、libgr.a の両方か又はどちらかが見当たらないからです。これは、grobot をインストールした人に場所を聞いて、-I, -L オプションをつけて回避して下さい。

もしきちんとコンパイルが出来たなら、実行してみましょう。

もしも、connection refused と言われてしまったら、grobotd を立ち上げてください。また、xgrobot を上げておかないと、robot をみる事が出来ません。これも上げておきましょう。

画面に sample と名前のついたおかしなキャラクタが表示されましたか? 表示されたのなら、プログラムはうまくコンパイルできました。では、sample プログラムを、&をつけて二つ実行してみて下さい。

二つ表示されたとたんに、両方で弾を撃ち合い始めましたね。そしてどちらかが負けてしまった事でしょう。その場を動いていさえすれば、やられなかったのに... ということで、なぜこのような動きをしたのか、このプログラムを上から見ていきましょう。

```
3 #include "sample.bitmap"
```

まず、16 × 16 の bitmap を#include を用いて読み込んでいます。これは、19 行目でsample_bits として使っていますね。

```
5 #define RADARMAX 8
6 SCAN radar[RADARMAX];
```

grobot.h で定義されている、SCAN 型の変数をRADRMAX 個だけ確保していますね。SCAN 型は、索敵の時に使われます。

```
19 id = gr_init( "sample", argc>1?argv[1]:NULL, '*', sample_bits );
```

grobotd に接続するための関数、gr_init を使用しています。ここでは、自分の名前を “sample” にして、argv[1] があるなら、そこのホストへ接続し、キャラクタ端末での自分を、‘*’ とし、sample_bits で定義されている bitmap を登録しています。また、帰り値を使用してid を読み込んでいます。

```
21 n = gr_scan(radar,RADARMAX);
```

索敵用の関数である`gr_scan`を呼んでいます。ここでは、`SCAN`型の`radar`に`RADRMAX`ぶんだけのデータを読み込むよう指示しています。そして、実際に読み込まれた数を`n`に保存しています。

```
23 gr_missile(radar[1].x,radar[1].y,4);
```

攻撃用の関数である`gr_missile`を使用しています。ここでは、`gr_scan`で読み込まれた敵の一番目の現在置に向けて強さ 4 のミサイルを発射しています。

これらの事をみて、`mkgrobot.*`に書いてある通り、このプログラムは敵をスキャンしてその場所にミサイルを撃ち込むという作業しかしていません。これでは、移動するわけがありませんね。

3 grobot で使用できる４つの関数

grobot では、`gr_init`, `gr_scan`, `gr_missile`, `gr_thrust` の 4 つの関数が用意されています。ここでは、その 4 つの関数の働きをみていきましょう。

では、grobot で使用できる関数を解説します。まだまだまともなプログラムは作れませんが、この章を読めば、grobot のプログラムは全部組める事になります。頑張っていきましょう。

3.1 gr_init

grobotd に接続するための関数です。この関数は、次のように定義されています。

```
int gr_init( robotName, hostname, appear, bitmapData )
 char *robotName, *hostname, appar, *bitmapData;
```

`robotName`は、自分のロボットの名前を入れます。これは、xgrobot で表示される名前になるので、カッコ良い名前をつけましょう。

`hostname`は、grobotd の動いているホストを書けば OK です。NULL が指定されると、localhost に接続しようとします。

`appear`は、キャラクタ端末用の文字を一文字だけ指定します。これは、まだキャラクタ端末用のビューワが出来ていないので、今のところは意味の無い物ですが、とりあえず要求されるので、書いておきましょう。

`bitmapData`は、bitmap(1) コマンドで作られる形式の bitmap を入力します。実際には、こんな形になっています。

```
#define nerv_width 16
#define nerv_height 16
static unsigned char nerv_bits[] = {
   0xc2, 0x0f, 0xe2, 0x07, 0xe4, 0x03, 0xf8, 0x3f, 0xf0, 0x7f, 0xe0, 0xff,
   0xc0, 0xff, 0xe9, 0xdf, 0xeb, 0x3f, 0x2d, 0x7e, 0xe9, 0x7c, 0x00, 0xf8,
   0xe0, 0xf4, 0x20, 0xf5, 0xe0, 0xd4, 0x20, 0x89};
```

`gr_init`は、登録されたあなたの robot の id を返します。また、`gr_init`がエラーを返す事はありません。エラーが起きると、exit するように作られています。

3.2 gr_scan

索敵用の関数です。`gr_scan`は、以下のように定義されています。

```
int gr_scan( bufP, n, n_returnP )
 SCAN *bufP;
 int n;
```

butP は、(SCAN *) 型の変数です。ここで指定した変数に、結果が入って帰って来ます。そして、一番最初の要素 (butP[0]) は必ず自分のデータが入ります。

n には、scan する robot と弾の数の合計を指定します。(もちろん、ここの数を、butP に指定した変数で確保できる以上の数を指定してはいけませんよ) また、もしn を 0 以下の数にした場合、n は 1 として処理されるようです。

gr_scan は、実際に scan された robot と弾の数の合計を返します。もちろん、自分の数も入っているので、必ず 1 以上の数が帰って来るわけです。

SCAN 構造体は、次のように定義されています。

```
typedef struct {
 g_type obj;/* SC_RBT または、SC_MSL のどちらかが入ります */
 g_id id;/* grobot または弾 の ID 番号が入ります。 */
 g_cord x, y;/* grobot または弾 の x,y 座標 */
 g_cord vx, vy;/* grobot または弾 の x,y 方向の速度 */
 g_cnt damage;/* grobot の ダメージ (弾の場合は、常に 0 )*/
} SCAN;
```

実際は、g_type, g_id,g_cnt は、int 型、g_cord は、double 型です。

3.3 gr_missile

攻撃用の関数です。gr_misslie は、以下のように定義されています。

```
int gr_missile( targetx, targety, misc, remain )
  double targetx, targety
  int misc, remain;
```

targetx, targety は、ミサイルを爆発させる場所を指定します。ミサイルはここまで飛んで行って、そこで爆発します。もちろん、爆発するまでミサイルは全く無害です。

misc は、ミサイルの強さを決定します。強さは、1.0〜MAXMISSILE までです。ちなみに、初期状態では、MAXMISSILE=8.0 となっています。

3.4 gr_thrust

移動用の関数です。gr_thrust は、以下のように定義されています。

```
void gr_thrust( hori, htime, vert, vtime )
 double hori, vert;
 int  htime, vtime;
```

hori, vert は、左右、上下の加速力をいれます。hori は、右が正、vert は、下が正です。この値の絶対値は、AMAX を越えない事が条件です。デフォルトのAMAX は、24.0 です。

htime, vtime は、左右、上下の加速力をどのくらい持続するかの時間を指定します。この値を 0 にすると、与えられたパワーで加速し続けます。

ようするに、もし、左に 20、下に 40 の速度を持っていたとしたら、

```
gr_thrust(20.0 , 1 , -20.0 ,2);
```

と実行すれば、ピタリと止まる事が出来るのです。

また、移動していて壁にぶつかってしまうと、ダメージを受けてしまいます。気をつけましょう。

4　sample プログラムを改造してみよう

さて、sample プログラムは、ただの固定砲台、それも敵を狙う事さえしないという、まことに頭のいたいプログラムでした。ここでは、sample プログラムを改造して、とりあえず敵の移動する位置を予測して弾を撃つ形に改造してみます。

敵の位置を予測するのに、$s = vt$ の式を利用して、敵の移動先を予測します。ここで不明なのが、vと t の二つです。vは、gr_scan で情報を得る事が可能ですね。t については、弾が到達する場所までの距離を弾の速度 (実際は 7) で割った物が答えなのですが...弾の到達する場所というのは、どこなのでしょうか? これはちょっと難しい問題なので、とりあえず現在の敵の位置で代用する事にします。では、プログラムを書いてみましょう。

```
#include <stdio.h>
#include "uirou_grtool.h"
#include "sample.bitmap"

#define SCAN_MAX 2
SCAN scan[SCAN_MAX];

void main(void){
  double t,guess_x,guess_y;

  gr_init("lec1", argc>1?argv[1]:NULL,'l',sample_bits);
  while(1){
    while(1 < gr_scan(scan,SCAN_MAX)){
      t = sqrt((scan[0].x- scan[1].x) * (scan[0].x- scan[1].x)
               + (scan[0].y- scan[1].y) * (scan[0].y- scan[1].y))/7.0;
      guess_x = scan[1].x + scan[1].vx * t;
      guess_y = scan[1].y + scan[1].vy * t;
      gr_missile(guess_x,guess_y,8);
      time1 = gr_time;
      memcpy(&scan_tmp,&(scan[1]),sizeof(SCAN));
    }
  }
}
```

このプログラムの重要な点は、

```
      t = sqrt((scan[0].x- scan[1].x) * (scan[0].x- scan[1].x)
               + (scan[0].y- scan[1].y) * (scan[0].y- scan[1].y))/7.0;
```

と、

```
      guess_x = scan[1].x + scan[1].vx * t;
      guess_y = scan[1].y + scan[1].vy * t;
```

の二点だけです。これは、スキャンした時点での自分と敵までの距離を、弾の速度 (7) で割り、それを弾の到達時間とし、その時間をもとに、弾の到達地点を予測します。

これで、きちんとではありませんが、敵の位置を予測するようになりました。今度は、移動して敵の攻撃を避ける必要がありますね。でも、その点は、皆さんで考えてみて下さい。

5 grobot での戦略

grobot では、このシンプルな中に奥の深い戦略の要素があります。ここでは、その戦略について、ちょっとだけ考えてみます。

まず、自分の robot が勝つためには、相手を倒さねばなりません。これは、相手の移動先を予想して、弾を撃つ事につながります。今回のレクチャーでは、敵と弾を発射した時点での敵の距離が変わらないものとした予測プログラムを作ってみましたね。

そして、少し考えてみると、敵との距離が短くなれば短くなるほど、先程の予測はより正確さを増すことが解ります。そこで、ただ近寄れば良いかと言うと、そうでもありません。自分も爆風に巻き込まれてしまう可能性があるのです。

そこで、敵のと距離が縮まったら、爆風の少ない攻撃をする、つまり、パワーを敵との距離でコントロールしながら戦うプログラムが考えられます。

では、次に、移動する事について考えてみましょう。移動する事は、直接自分の robot を破壊から守る事につながります。これをしないと、`sample.c` のようなクソプログラムにも簡単にやられてしまいます。(`lecture.c` も、そうですね) そこで、まずは回避のために、いつも動き回る事が肝心です。

では、ただ動き回るだけで良いのでしょうか? それはそうとも限りません。例えば、`lecture.c` のようなプログラムと戦う事と仮定しましょう。この場合は、`lecture.c` は加速度を考慮にいれていないので、いつでも加速して他の方向に行く避け方と、位置予測アルゴリズムの盲点である、距離が変化するとどうしようもないという点を利用して、近付いたり、遠ざかったりするプログラムが考えられます。他にも、ただただ加速して、素早い動きで敵を翻弄するプログラムなどが考えられますね。

6 grobot の仕組み

grobot は、サーバークライアント形式で動作しています。実際には、grobotd が、全てのロボットの動作を管理するサーバーで、grviewd は、robot の動きを表示するための xgrobot のためのサーバーです。図解するとこんな感じになります。

```
       grobotd <--------------> grviewd
       / / | \                  / |  \
      / /  |  \                /  |  xgrobot
 robot /   |  robot       xgrobot  |
      /    |                       |
   robot  robot                  xgrobot
```

grobotd は、画面に関するデータを grviewd に送るだけでよく、grobotd 自体の高速化を計っています。この図の robot にあたるプログラムを我々が書くことになります。

実際、ネットワークに関する事項は、全て libgr 上で行われているので、ユーザーは気にする必要はありません。

ということで、grviewd とお話すれば... 以降自主規制。

7 bitmap の使い方

Xgrobot では、bitmap を用いて自分の robot を表示します。ここでは、その bitmap の作成用ツールである、bitmap(1) について解説します。

bitmap(1) は、X のプログラムで、`/どこそこ/X11/bin/` にあります。これの使い方は簡単で、まず起動すると、16×16 の桝目が現れます。ここに、マウスの 1 ボタンで黒をセットしてマウスの 3 ボタンで白をセットしていきます。Xgrobot では 16×16 のビットマップ絵を必要としていますので、このまま絵を書いて、File メニューから save as を選び、ファイル名を指定して、セーブすれば OK です。例えば、`sample.bitmap` と名前をつければ、そのファイルは`sample.c` で使用可能になります。

8 grobot13 の仕様

grobot には、知っておくとちょっと便利になる数値がいろいろあります。ここでは、ソースファイルなどを探っていて、ちょっと便利かも知れないと思ったデータを挙げておきます。だたし、このデータは grobot のバージョンアップとともに変更される可能性もあるとの事を覚えておいて下さい。

ミサイルの速度は 7.0 です。`grobot.h` で`M_SPEED` と、定義されています。

`gr_init()` で登録できる自分の名前は、32 文字までです。`common.h` で`MAXNAME` と、定義されています。

grobot の最大加速度は、x,y 各方向に 24.0 までです。`grobot.h` で "AMAX" と、定義されています。ということは、斜め方向なら 34 近くまで加速出来ると言う事ですね。

フィールドの大きさは、512×512 の大きさがあります。`grobot.h` で`FIELD_[WH]` として、定義されています。

grobotd には、最大 8 機の robot が接続できるようです。これは、`common.h` で`MAXROBOT` として定義されています。

ロボットは、12 以上のダメージを受けると、壊れてしまいます。`grobot.h` で`DEADDAMAGE` と、定義されています。

grobot のサイズは 16×16 です。この範囲に爆風がくればダメージを負います。

爆風の広がる時間は、そのまま missile のパワーによるようです。パワーの値と同じだけ爆発は持続します。

爆風が広がるのは、(`M_POWER*mP->now`) の式で決定されています。これから、爆風は 1 ラウンドごとに、8 づつ半径を大きくしているようです。

robot のダメージは、1 づつしか上がりません。これは、爆風に巻き込まれていたラウンド数だけダメージを食らうと言う事です。

壁に当たると、1 のダメージを食らい、当たった軸の速度が逆転します。

Part IV
ぐらふぃてぃ

家庭用ゲーム機エミュレータに関する覚書

uirou@mma.club.uec.ac.jp

1　事のはじまり

それは、にふてぃとかいう大手パソコン通信会社に接続したある日の事だった。その中の一人の書き込みが僕の心を大きく動かした。要約すると、こうだった。「Windows で走るファミコンエミュレータなるものが、とある ftp サイト[20]に転がっているらしい」便利な事に、僕ら電通大生は学校の端末を使い ftp をただで出来るという環境にいた。もちろん、僕はその環境を利用し早速そこに行ってみた。

そこに行ってみると... 遅い。返事が帰って来ない... あ、来た。どれどれ...

Connection timed out.

ぐはっ! コネクションがはれない!くっそー。そんなに遠いのか。まぁ、いいや。もう一度つないじゃえ。えい!...

Connection timed out.

ち... もう一度、えい!

```
User anonymous access denied.
This means that:
    An FTP protocol error occurred.  Please try again.
Remote server replied with:
Sorry, the site is currently full (15 out of 15 users)
```

くすん。だめか。

てな感じで、何度もつないだ所、そこには NES[21]の他にもたくさんのエミュレータがある事がわかってきた。それはすなわち...

Gameboy, GameGear, SEGA MarkIII, SEGA Mastersystem, Genesis, Amiga, MSX...

数え上げれば切りが無い。何と素晴らしい! どれもこれもゲーム機ばっかりじゃないか! [22]これは...

ぜーんぶ持って来るしか！

ということで、その日から僕の「エミュレータ猿」生活が始まった。

2　エミュレータ、ああ、エミュレータ、エミュレータ

さて、世の中にはいろいろなエミュレータが存在するが、僕が現在猿になっているエミュレータは、簡単に言って、家庭用ゲーム機のエミュレータだ。その他のエミュレータに関しては、全く調べていないので、ちょいと良く分からない。

エミュレータは基本的に ROM が無いと動かない。ROM というのは、この場合、ファミコンなどのカセットの ROM イメージのことだ。要するに、エミュレータがファミコン本体で、ROM がカセット、と言う事だと思ってもらえれば、OK だ。もちろん、カセット内に RAM を持っててセーブする機能のあるものでも、きちんとエミュレートしてくれて、`hoge.sav` とかいうファイルにして残してくれる。もちろん、オーディオデバイスがあれば、音だって、かなりきれいに鳴る。うまく行けば、十字キージョイスティックを使用して、本物と同じような感覚で楽しむ事も出来る。

[20] ftp://alpha.pulsar.net/pub/Stumble/Other/NES/
[21] にんてんどーえんたーていんめんとしすてむ。要するに日本で言えば、ファミコン
[22] Amiga や MSX はゲーム機じゃないって? これは失礼。でも、まぁ、いいじゃん。他はゲーム機なんだし

しかし、エミュレータを作っている暇人[23]たちは、大抵が外国の人達だ。もちろん、その ROM を吸い出しているのも外国の人達だ。ということは、要するに日本語に関しては、なーんも考えていない事が多いのだ。

でも、まぁ、エミュレータの方に関しては、漢字フォントを標準装備している家庭用ゲーム機なんて多分ありえないので、エミュレータのほうは日本語化されいなくても、大体問題にはならない。

しかし、ROM の方は、日本語の ROM が落ちている事は少ない。実際、落ちている ROM のほとんどは、英語圏の ROM であり、日本の ROM の場合、わざわざ jap[24]とか書いてあったりする。英語圏の ROM で、悲しくなってしまうのが、RPG や AVG などの文字が重要なウェイトを持っているゲームだ。この手のは持って来ても、あんまり嬉しくない。結局ああ、こんなシーンもあったっけなぁ、ってな感じで「ぽい」である。しかし、Genesis[25]の ROM のように、日本のモード、アメリカのモード、ヨーロッパのモード、といくつかのモードを同じ ROM にいれておける機種の場合は、その点は全く問題無しになったりもする[26]。

さて、エミュレータは、かなりいろいろなプラットホームで動いている。大体の能力を書いてみると、

	速さ	音	ジョイスティック
MS-DOS	とても速い	普通付いてる	普通対応
Windows	かなり速い	付いているかも知れない	多分対応
UNIX + X	遅い	付いているかも知れない	多分非対応
Mac	未確認	未確認	未確認

てな感じである。ただし、NES には、Linux+SVGAlib のバージョンが存在する。その場合、大体、MS-DOS 並のスピードが出る事が確認されている。

てなわけで、エミュレータを動かしてまともに遊ぼうと考えた場合、

MS-DOS を使うのが一番いい

ようである。

では、それぞれのエミュレータについて少しづつ解説してみる。

3 現在発見されているエミュレータのそれぞれ

Amiga

uae

Plathome: MS-DOS(!98), UNIX+X, Linux+SVGAlib

HomePage: UNKNOWN

Support: Sound, Joystick

Comment: Amiga の、68020version のエミュレータらしい

Colco

colem

Plathome: MS-DOS(!98), UNIX+X, Win32+WinG

HomePage: http://www.freeflight.com/fms/ColEm/
ftp://ftp.komkon.org/pub/Coleco/
ftp://altair.komkon.com/pub/Coleco/

Support: Sound, Joystick, autofire

[23] それで遊んでいる僕も暇人 (自爆

[24] これって、やっぱりジャップって読むのかなぁ

[25] MEGADRIVE のことね

[26] それでも、RPG などの文字が多いゲームは、“USE ONRY GENESIS SYSTEM” とかいわれちゃって、日本モードじゃぁ動かなかったりするけれど。

Comment: Colcoっていうゲーム機らしいけれど...

GameGear

MasterGear

Plathome: MS-DOS(!98), UNIX+X, Mac, MS-Windows

HomePage: http://www.freeflight.com/fms/
http://consider.ferris.edu/ santino/game-gear/games/

Support: Sound, Joystick, autofire

Comment: ほぼ完璧な GameGear エミュレータ。速度も本物と同じくらい出る。ちょっと気になるのは、音が少しだけ違う事。同じエミュレータで、Master System もサポートしている。[27]

GameBoy

Virturl GameBoy

Plathome: MS-DOS(!98), UNIX+X, Amiga, Mac, MS-Windows

HomePage: http://freeflight.com/fms/VGB/
http://www.komkon.org/ dekogel/vgb.html
http://mail.bcpl.lib.md.us/ dstrickl/gbgames/

Support: Sound(!UNIX), Joystick, autofire

Comment: かなりのレベルの GameBoy エミュレータ。まだたまに落ちる ROM がある。UNIX 版は、画面が小さすぎる。

Genesis

GenEm

Plathome: MS-DOS(!98)

HomePage: http://www.htw.uni-sb.de/people/mgietzen.html
http://149.152.46.141/liston/genesis/
http://198.114.144.146/emu/

Support: Sound(!FM-sound), Joystick(?), japanese mode

Comment: あの MEGADRIVE のエミュレータ。現在、Ver.0.15 まで出ている。さすがに Ver.0.15 だけあって、完璧な動き方はしてくれない。音もでない物が多い。Joystick は、サポートされたと書いてあるのだが、家にあったものでは動かなかった。シャイニングフォース 2 が動いて、幸せー。

Genesis Emulator

Plathome: MS-DOS(!98)

HomePage: UNKNOWN
http://149.152.46.141/liston/genesis/
http://198.114.144.146/emu/

Support: UNKNOWN

Comment: 動かしたけれど、なぜかうまく動かない。よって何もわからない。でも、GenEm が動いたから、いいや。

MSX

fMSX

Plathome: MS-DOS, UNIX+X, Mac(!68k), Amiga, MS-Windows

[27] って、ハード的には Z80 を使っていたりして、ほとんどおんなじ物なのだから、当然ではある

HomePage: http://www.freeflight.com/fms/fMSX/
ftp://altair.komkon.com/pub/MSX/
ftp.saitama-u.ac.jp:/pub/msx/fMSX/

Support: Sound, Joystick

Comment: 非常に出来の良い MSX エミュレータ。MSX から MSX2+までサポートしているらしい。最近、音も出るようになって、どんどん完璧になっている。公式 ftp サイトが日本にあるので、利用すべし。

SEGA Master System & SEGA Mark III

Master Gear

Plathome: MS-DOS(!98), UNIX+X, Mac, MS-Windows

HomePage: http://www.freeflight.com/fms/
http://consider.ferris.edu/ santino/sms/games/

Support: Sound, Joystick, autofire

Comment: ほぼ完璧な Master System エミュレータ。速度も本物と同じくらい出る。ちょっと気になるのは、音が少しだけ違う事。同じエミュレータで GameGear もサポートしている。

Nintendo Entertainment System

iNES

Plathome: Linux, Linux+SVGAlib, FreeBSD, OSF1(Alpha), Solaris(sperc), MS-Windows+WinG

HomePage: http://freeflight.com/fms/iNES/

Support: Sound, Joystick

Comment: あのファミコンのエミュレータ。他のエミュレータも、おんなじ人が作っているみたいなのに、何故かこれだけは、バイナリ配布になっている。しかし、Linux+SVGAlib はかなりサクサク動いてくれる。非常に良くできたエミュレータである。この後出て来るパソファミと張り合っているらしい。Windows 版はシェアウエアであり、レジスト前は、音が出ないなど、制限事項が多い。

パソファミ

Plathome: Win32+WinG

HomePage: UNKNOWN(perhaps Niftyserve)

Support: Sound, Joystick

Comment: Windows 版だけしか作られていない。作者は日本のにふてぃとかいう所に生息しているらしい。にふてぃの Windows ソフトの例に洩れず、儲けるつもりでお金を取るシェアウェアらしい。レジストしないとゲーム内時間で 3 分しか遊べない。[28]作者さんがにふてぃで言っている事によれば、「この頃勝手にコピーして、インターネット上でばらまいているふとどき者がたくさんいます。みなさんはそのような所からはダウンロードしないでください。」という事らしい。

PlayStation

PSMooSim

Plathome: Amiga, Solaris(sparc)

HomePage: http://stekt.oulu.fi/ flame/
ftp://x2ftp.oulu.fi/????/

Support: UNKNOWN

[28]実は、レジストされたバイナリが出回っているという話もある。(まぢ

Comment: あの、PlayStation のエミュレータのようである。
でも、“SIMULATOR/MONITOR” と書いてある。エミュレータではないのだろうか? とにかく、バイナリ配布で、Amiga と Solaris 用しか無い。よって確認はしていない。まぁ、Amiga にしても、Solaris にしても、プレステとテレビを買った方が、安そう。

Super Nintendo Entertainment System

ViRTUAL SUPERMIGiCOM

Plathome: MS-DOS(!98)

HomePage: http://www.iceonline.com/home/thebrain/vsmc/
http://www.futureone.com/ damaged/Consoles/SNES/files/

Support: Sound, Joystick

Comment: マウスドライバが必要。かなり良い評価を得ているようなので、良いエミュレータのようだ。

Esnes

Plathome: MS-DOS(!98)

HomePage: UNKNOWN
http://www.futureone.com/ damaged/Consoles/SNES/files/

Support: UNKNOWN

Comment: DOS で走る SFC エミュレータらしいのだが、まだ動かした事が無い。よってなんにもわからないが、Ver.0.02b らしいので、あんまり期待しない方が良いかも知れない。

スーパーパソファミ

Plathome: Win32+WinG

HomePage: UNKNOWN(perhaps Niftyserve)
http://www.futureone.com/ damaged/Consoles/SNES/files/

Support: UNKNOWN

Comment: パソファミと同じにふてぃの作者さんのスーパーファミコンエミュレータ。これもシェアウェアなので、レジストしないと試用中はゲーム内時間で 3 分しかない。[29]で、ROM が無いのでチェックしていない。

4 その他のエミュレータ関連の HomePage

Node99

Location: http://www.nfinity.com/ swhalen/node99/

Comment: 僕が毎日のようにお邪魔しているエミュレータのサイト。エミュレータの能力を星の数 (五つが最高) で評価してあるので、ひどいものを掴まされなくてよい。また、頻繁に更新されているので、それぞれの HomePage に行かなくても、新しいバージョンが出た時などは、ここに行くだけで OK.

Archaic Ruins: Damaged Cybernetic's Emulation Site

Location: http://www.futureone.com/ damaged/AR/

Comment: かなり整備のゆきとどいているサイト。Node99 と、ここに行っていれば、あとは ROM の心配をするだけで良い。[30]

ftp://alpha.pulsar.net:/pub/Stumble/Other/

[29] これにも、レジストを解除するためのバイナリや、英語化パッチなどが出回っているという。(まぢ
[30] ROM の ftp サイトにリンクが張られてはいるのだが、まともな ROM が置いていない場合が多いようだ。

Location: ftp://alpha.pulsar.net:/pub/Stumble/Other/

Comment: 全ての事の始まり。ここにはありとあらゆるゲームのエミュレータがある。でも、無茶苦茶遠いし、ほとんどつながらない。つながっても、90 秒レスポンスが届かないと切られてしまう... ち。[31]

Emulation on the Mackintosh

Location: http://www.komkon.org/ stiles/emulation/

Comment: Mac の人のためのエミュレータのサイト。Mac はよくわかんないや。

ftp://ftp.komkon.org/pub/EMUL8

Location: ftp://ftp.komkon.org/pub/EMUL8

Comment: ちょっと遠いんだけれど、エミュレータを置いてある。(Rom は無いみたい)

EMULATOR ZoNE

Location: http://www.geocities.com/SiliconValley/park/2912/index.html

Comment: ここも各種エミュレータが置いてある。でも更新はあまり早くはないみたい。

Emulation Zone

Location: http://www.tiac.net/users/wacko/emulator/emulzone.htm

Comment: 最近みつけたエミュレータのサイト。Megadrive+MegaCD や、Megadrive+32X 等の事についても書いてあるので、見てみるといいかも。[32]

Genesis and Super Nintendo Games

Location: http://149.152.46.141/liston/games.htm

Comment: 珍しく、ROM が置いてあるサイト。SEGA な人で、エミュレータ猿は行くべし。

5 Your own risk

さて、エミュレータを使用する時には、普通 ROM を使うわけではあるが、実際、売り物であった ROM を使用する事は法に触れる。ということをお忘れ無く。

[31] ここには腐る程 ROM が転がっている。できれば全部 get したいが...
[32] でも、MegaCD のバイナリは、なくなっているみたい...

MMA 人物録—現役編

Nakahara Takafumi <ottan>

ちょっと気が向いたので私見にもとづいて、おもな部員たちについて一筆書きたいと思う。

はじめに断っておくが、この文章は、あくまでも “私見” つまり独断と偏見、自分の印象にもとづくものであり、なんら客観性を主張するものではない。

tree: MMA の偉いひとその 1。

ほんとにいいひと。

UNIX 一般、ハードウェア、など余人の及ばない域に達している。

しかし、昨今のあまり高尚[33]とはいえない趣味に傾斜してしまったため、壊れただの、人間かわっただの、種々吹聴されるに甘んじてらっしゃるようである。

pman: 最近、AIH で苦労されているようですね。

matuzaki: アスキーバイターその 1。

スーパーアスキーで、分室 LAN の構築、新製品のレビューと活躍されているようである。

p2: 宇宙研に詰めてらっしゃるので、あまり部室で見かけないですね。

TRON のひと。

shiget-o: 温泉のひとその 1。

交友関係、趣味ともに驚くほど広い人である。

現役 MMA でもっとも精神年齢がふけてるひと。

imai: 三度の飯より、論争の好きな人。

しかし、最近、みんな相手になってくれないので、物足りないようである。

基本的にいいひと。あくまで “基本的に” だが、

shimizu: いろんな意味ですごいひと。うでのいいハッカーにして破壊魔。

MMA 部室内で破壊されたもののうち、八割方はこの人の手によるものではないだろうか。

最近、長年愛し続けた X68k から AT に乗り換えたようである。

“おごってもらうと高くつく” とは、某佃先輩の弁である。

uesakasan: MMA ではめずらしく、おとなしそうで、まじめなひと。

しかし、しれっと home の下には crack などという名前のディレクトリが掘ってあったりするそうである。

ムーミンとおともだち。

rikachan: ぼくから申し上げることはなにもございません。

tree さんとは別の意味でも、よぢんの達しえない域にいっちゃってますよね。

Diaw: いまいさんに、“神が存在するという証拠はない” ということにさせられてしまったひと。ひどいね、いまいさん。

okada: イベンター。

きはいいが、基本的にひどいひと。

[33] ここで言う高尚とは “人前で話してもはずかしくない” というていどの意味である

takkun: ういろうさん。

立場の強くはない現会計。

route: 現部長。

sanjigen いわく “自分から不幸になってるよね”。

形式や手段にこだわりすぎて対人インターフェイスは不調である。

しかし、プライドが高く、人付き合いは嫌いだといってる割には、いいひとである。

curry: 自称ゆうじんおもいいのいいひと。

だだ、ちょっと、ルーズなだけ。

kyougoku: 臑に傷を持つ男。

MMA 部員の中では比較的まともにみえが、あくまで見えるだけ、という話もある。

toyama: MMA にいるのが不思議だ、と言われ続けた、ほんとにいいひと。

時々ナイスなボケをとばす。

oolong: モスバーガーが好物。

ここのところ、IIJ が忙しいようですね。

so-miya: 話の振り方がちょっと下手な以外はまじめそうで、いいひと。

harada: アスキーバイターその 2。

アスキーバイトの経験のない人以外は、real なはらださんを見たことある人は少ないんじゃないかな。

eggman: 口数はすくないけど、大崎さんや京極さんに次いで、AT と秋葉にくわしいんじゃないかな。

sanjigen: きらいなたべもののおおいひと。

ottan: 筆者。MMA では、つっこみが鋭いというか、口が悪く、態度がでかいということになっている。1 年のときは部室のぬし状態だった。

etc: 新人のひとや、ゆうれいなひとはよーわからんのでパス。

総評

こうして、全体を通して見てみると、いかにまともな人が少ないかわかる。

近年、特にアニメ関係への傾倒が著しいようだ。以前からその傾向はあったうだが、最近では部室まで持ち込んでしまっているため、よけい顕著に感じるようである。

しかし、麻疹レベルで終わればいいのだが、ぼくにはどうも、死に至る病に思えてしかたない。

まぁ、このへんはおもしろおかしく書いてる部分なのでどうでも良いことであるが、しかし、ある程度偏りのある集団[34]であることは否めない事実であろう。もうすこし人材の多様性があってもいい気はする。(確かにある意味では多様だとは思のだが...)

ということで、“種の多様性が繁栄をもたらす” ということばを信じるならば、MMA インカレ[35]化プロジェクトとか起こすとたのしいかもしれない。

文責は、ottan こと中原隆文にあります。

[34] そもそも電通大と言う母集団からして偏ってるという話もある

[35] カレッジサークルの略、母集団となる大学が複数あるサークルのこと。

MMA に IP が来た日

佃 良生 (route@mma.club.uec.ac.jp)

「アイコブ総統より電信です, 本日 17:00 までにトレマイン艦隊へのパケット到達を行うこと. それ以降の接続には支援出来ないそうです. 」

艦長 「現在の接続状況は? 」

「現在第 4 段階です. 終了時刻まであと 640 秒, ケーブル敷設の際のトラブルのため, 大幅に遅れています. 」

艦長 「総員第一種戦闘配備. 波動エンジン全開. 17:00 までにトレマイン艦隊へのパケット到達を確保せよ. 」

「機関室より報告です. 現在波動エンジン 出力 120%. 30 秒で臨界点に到達します. 」

艦長 「臨界点直前まで続けろ, 予定終了時刻は? 」

「予定終了時刻まであと 380 秒, アイコブ総統の指定時刻ぎりぎりです. 」

「機関室より報告. エンジンにトラブルが発生, 波動エンジン出力 40%まで落ちています, 予定終了時刻は,... だめです間に合いません. 」

艦長 「10Base-T を司令席へ回せ, 総員持ち場にて待機. 」

「艦長, 危険すぎます. 艦長のコネクタは RS-232C です. 10Base-T には毎秒数千のパケットが,... 無理です. 」

艦長 「いいから回せ! 」

「しっ, しかし... 」

艦長 「しかしもなにもない 私が MMA のみんなにしてやれることはこのくらいだ, あとは君に任せる. いいから回せ! 」

「てっ10Base-T 生体コネクタへジャックイン, 現在の実測値 10M bps. 理論値の限界に達しています. 」

艦長 「ぐわぁぁぁ 」

「艦長! 」

艦長 「大丈夫だ, 君は報告を続けたまえ. 」

「現在の実測値 10M bps あと 15 秒で 終了です. 13, 12, 11, 」

「4, 3, 2, 1. トレマイン艦隊への接続を確認. IP によって交信出来ます. 艦長! やりました. 」

「艦長! 大丈夫ですか? 」

艦長 「大丈夫だ. これで MMA とトレマイン艦隊とで共同戦線をはることができる. この一歩は MMA にとっての大きな一歩だ. あとは君に任せたぞ... うっ」

「艦長? 艦長ぉぉぉぉ 」

こうして MMA と トレマイン艦隊との IP 接続が行われた. そして MMA 艦長楯岡孝道は 静かにまぶたを閉じるのであった.

この物語は事実を基にしたフィクションであり, 実在の人物, 団体とは一切関係ありません. ましてや宇宙戦艦ヤマトとは何の関係もありません.

部長の傾向と分析

sanjigen

Abstract

MMA における部長の役割とは何か。その実態は。ここに、MMA の部長としての役割を問い求め、疲れた一青年が登場する。さあ、パロディを交えながら考察してみよう。

なお、本記事は人気 TV アニメ “機動戦艦ナデシコ” 第 5 話とは関係ありません。

佃[36]「はぁ...」

sanjigen[37]「佃さん、あたま[38]。」

佃 「はぁ... 疲れた。sanjigen、部長っていうのは何なんだ。」

sanjigen 「知りたいですか?」

佃 「ああ、知りたい。」

sanjigen 「教えることはできなくてもデータなら...」

佃 「へ?」

sanjigen 「思金、データ転送。ライブラリーより検索。部長の傾向と分析。」

佃 「sanjigen も成長したのぉ...」

^G[39]

sanjigen 「来ました。いつごろの部長にしますか?」

佃 「あ、ここ数年で。」

検索画面

sanjigen 「出ました。」

佃 「は。」

sanjigen 「最近の部長の傾向と分析。部室ネットワーク接続以降名部長と呼ばれた部長は出現していない。現在のように、部室がネットワークに接続され外部とのメールのやりとりに支障が無くなった時代は、部長ただ一人の決断は必要とされない。部長は外部との交渉の代表、ないしは部員の不平不満を気持ちよく吸収させる役目がとれれば良い。旧年は頼りになりそうなタイプもいたが、外部と摩擦を起こさないように、まめに対外工作をするタイプもいる。要するに現在は統率力や決断力などの本質的な意味での部長は必要としないのである。」

佃 「何だそれは? それって...それって...」

ottan[40] 「それって早い話、誰だって良いってことですね[41]。」

いじけながら退場する佃「もういい。余計疲れた...俺は調布祭が終わったら引退する。」

sanjigen 「中原センセイ[42]...」

[36]現在の MMA の部長である。人物評については本冊子の ottan の記事を参照のこと。
[37]MMA の書記。本記事の筆者。ottan の記事でも詳細は不明。
[38]最近、佃さんは頭をとってもリンスくさくしていたりする。
[39]“ピー!!” という beep 音を表す記号である。
[40]佃の天敵。実在する。
[41]ottan のツッコミの切れ味とタイミングは絶妙である。
[42]ottan の真実の名前なのさ。

ottan 「え? なに? ちょっといじめすぎた? (ニコニコ)」

sanjigen 「うん。」

ottan 「大丈夫。いつものことだから。すぐに立ち直るよ。(ニコニコ)」

　ううむ。肝心の “最近の部長の傾向と分析” がイマイチだな。でも、本当の佃さんはちゃんとマメに部長の仕事をやってくれていますよ。だから責めないであげてね。

1996年 新入生歓迎シーズン 百萬石

Special thanks	MMA 部員のみなさま
表紙デザイン	佃 良生
印刷	学友会室
編集	君島 秀征
発行	電気通信大学 MMA

1996年11月15日 発行

各投稿に関する意見, 疑問, 助言は、各投稿者へ。本冊子の編集, 組版に関する意見, 疑問, 助言は、君島秀征 (E-mail:sanjigen@mma.club.uec.ac.jp) までお願いします。