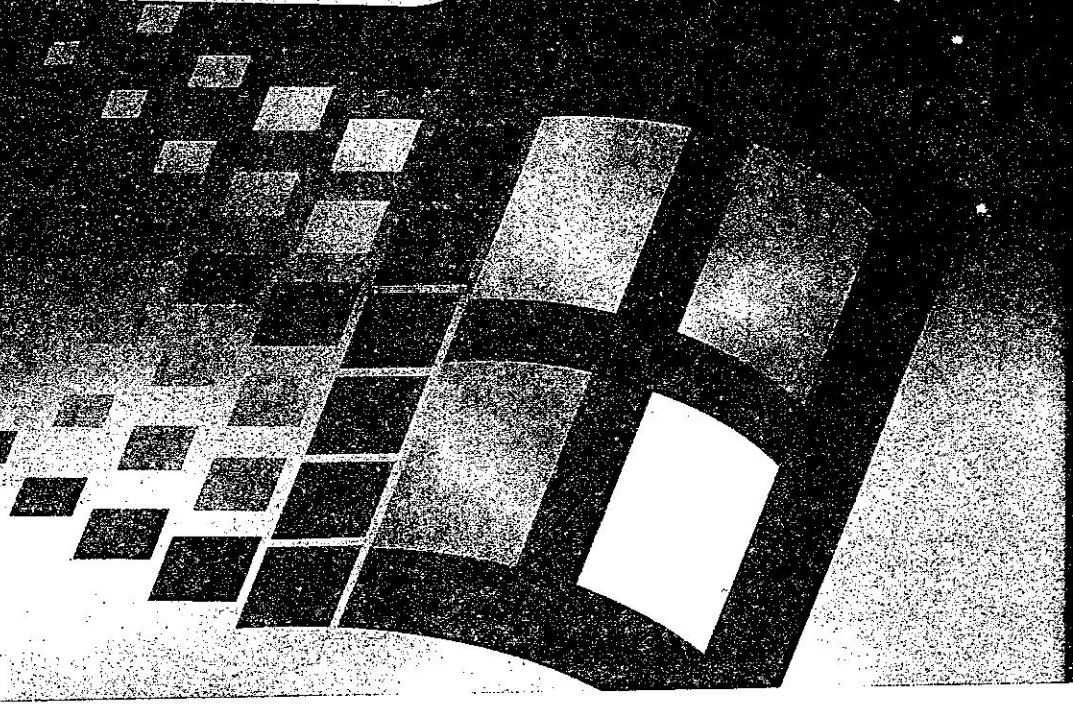
MMA

VERSION 4.0

Includes Microsoft Internet Explorer

CD-ROM

マイクロソフト ウィンドウズ NT ワークステーション オペレーティング システム



1998年度春

百萬石

新入生歓迎の言葉

ikidou@mma.club.uec.ac.jp

新入生のみなさん御入学おめでとうございます。MMA は Microcomputer Making Association の略であり、よりよいコンピューティング環境を実現するために設立されました。設立当初は文字どおりマイクロコンピュータを作成するサークルでしたが、ハードが発展した現在、ソフト作成等、各部員が興味をもっているものを自由に取り扱っています。

みなさんがこの百萬石を手にとり、読んでくださっているということは、コンピュータに対し何かしらの興味をもっている事と思います。それらの興味が一番大切なものです。現在の経験や知識なんて関係ありません。経験はつめば良いですし、知識は調べれば得ることができます。その原動力となるのが、「興味を持つ」ということです。

この百萬石を読んで、色んな意味で様々な興味がわいてくれば幸い[1] です。

最後に入部の有無に関わらずこの百萬石は大切に補完~H^H 保管して下さい。いつかきっと役に立つ時がくるはずです。

[1] さらに入部までしてもらえばもっと幸いです。

メーリングリストサーバ ARGO の改良
～ メタリスト機能の追加 ～

楯岡 孝道 (tree)
tate@spa.is.uec.ac.jp

平成 10 年 4 月

概 要

共通の話題が存在する複数のメーリングリストを利用している際に、共通の話題をどのように扱うかが問題となることがある。ARGO では、複数のメーリングリストのメンバーリストの和集合をメンバーリストとするメタなメーリングリスト (メタリスト) を作成することにより、共通の話題をこのメタリストで扱うことを可能にした。本文章では、このような問題の扱い方と ARGO による解決方法およびその問題点を述べる。

1 はじめに

複数の、共通の話題が存在するメーリングリスト (ML) を利用している際に、共通の話題をどのように扱うかが問題となることがある。

通常のメールではメールを関係する全ての宛先に送信すれば良いが、返信先 (Reply-to:) の付加などを行なっている ML の場合には、元メールに対する返信が単一の ML 宛になってしまう。この場合、関係する全ての ML に加入していないユーザは、話題を完全に追跡できなくなる。また、メールに通し番号が付けられていても、返信がどの ML で行なわれるか判断できないため、後に整理することが困難になる。

ARGO[1, 2, 3, 4] では、複数の ML (サブリスト) のメンバーリストの和集合をメンバーリストとする、メタな ML (メタリスト) を作成することにより、共通の話題をこのメタリストで扱うことを可能にした[1]。

2 共通の話題を扱う方法

複数の ML で共通の話題を扱うには、以下のような方法がある。

1. 全ての ML 宛に送信する
2. 複数 ML 中の、特定の一つの ML で扱う
3. その話題を扱うための新たな ML を作成する

1 の方法では、前節で述べたように返信に関しても明示的に全ての ML に出さなければならないという問題がある。また、関係する複数の ML に参加しているユーザには同じメールが複数到着することになる。

2 の方法では、1 のような問題は生じないが、共通の話題を扱う特定の ML に加入していないユーザが生じてしまい、全てのユーザに伝えたい情報は、結局全ての ML に送信する必要が生じてしまう。

3 の方法は、単純に新たな ML を作成した場合には、2 の方法と同じ問題が生じる。すなわち、共通の話題用の ML に加入していないユーザが存在するため、全てのユーザに伝えたい内容は、結局全ての ML に送信する必要が生じてしまう。

しかし、3 で作成する ML として、関係する全ての ML に加入している全てのユーザを重複無くメンバに持つ ML (メタリスト) を作成することができれば、関係する全てのユーザが加入している保証が得られるため、このような問題が解消される。単純に全 ML に対して送信したのとは異なり、メタリストに送信されたメールは、メールを受けとるべき全てのユーザに一通だけ到着することになる。

3 メタリストの設計

メタリストが効果的に動作するには、以下の条件が満たされる必要がある。

- 全てのサブリストの全ユーザを含まなければならない

[1] 現在の ARGO のリリースにはメタリスト機能は含まれていない

- 同一ユーザに複数通のメールを送らない

メタリストのメンバの管理を手動で行なった場合には、これらの条件を満たすメンバーリスト (メタ配送リスト) を維持管理するのは管理者にとって大きな負担となる。そこで、ARGO ではこれを各 ML のメンバーリストから自動的に生成することにより、管理者に負担をかけずにメタ配送リストを維持する。

1 メンバーリストの要素

ARGO システムにおけるメンバーリストは次の 4 つの要素から構成されている。

ID 番号	:	メンバーリスト中で一意に定まる数字
配送タイプ	:	メール配送形式とオプション
配送アドレス	:	メールの配送に用いるアドレス
別名アドレス	:	配送アドレスと同一とみなすアドレスの正規表現

ID 番号は、ARGO システム中での管理を容易にするために用意され、ダイジェスト配送の際の一時ファイルのファイル名などに用いている。そのため、この番号はユーザが ML の利用を継続している限りは同一のものを用いつづける。ただし、ユーザの脱退等で欠番になった番号は再利用される可能性がある。

配送タイプは、メールをどのような形式で送信するかを保持する。現在は、通常配送、MIME multipart/digest 形式配送、MIME multipart/mixed 配送形式、simple 形式がある。また、必要に応じて配送の一時停止を表すフラグ及び配送間隔を表わす数字を付加する。

配送アドレスはメールの配送に用いるアドレスで、そのままメールアドレスとして利用できる形式である。

別名アドレスにはメール配送は行なわないが、配送アドレスと同一とみなしてメールの受けつけ等を許可するアドレスを正規表現で記述する。これは複数記述できる。

2 メンバーリストの自動生成

メタリストでは、サブリストのメンバーリストに含まれるこれらの情報から重複のないメンバーリストを生成する必要がある。この生成にはいくつかの問題がある。以下にそれぞれの解決法を延べる。

1 メールアドレスの決定

メンバーは配送用メールアドレスと別名アドレスの組によって識別されるが、ユーザが全てのメーリングリストで同一の配送アドレスを用いているとは限らないという問題がある。

例えば、ある ML に次のように登録されているユーザがいると仮定する。

配送アドレス	`user@hostname.subdomain.domain.ac.jp`
別名アドレス	`user@subdomain\.domain, user@.*\.subdomain\.domain`

同時に、このユーザは別の ML で次のように登録されている可能性がある。

配送アドレス	`user@subdomain.domain.ac.jp`
別名アドレス	`user@.*\.subdomain\.domain`

この場合、これらのエントリは同一のユーザとして扱わなければならない。そこで、メンバの重複を確認する際には単順な配送アドレスの比較だけでなく、別名アドレスの正規表現が配送アドレスにマッチするかどうかも確認する。

2 ID 番号の決定

既に延べたように ARGO においては ID 番号は変化しない事が望まれている。単順にサブリストのメンバリストからメタ配送リストを生成した場合には、メンバの入脱退によって番号が変化する可能性がある。

そこで、メタ配送リスト生成時には過去に生成したメタ配送リストを参考として読み込み、メタ配送リストに存在するエントリに関しては ID 番号の再利用を行なう。これにより、新規ユーザ以外の ID 番号は不変となる事が期待できる。

3 配送モードの決定

ARGO には複数の配送モードが存在し、ユーザ毎に自由に配送方法を変えることができる。メタ配送リストの配送方法の決定は単順にメタ配送リスト生成中最初に見つけた配送方法を採用する。ただし、一部の ML で配送が抑制されている場合には配送が許可されている配送方法を採用する。

4 プライベートリスト

メタリストの基本構想とは相反するが、メタリストの配送だけを受けたくない、もしくはメタリストの配送だけを受けたいというユーザのために、メタリストにもそれ自身のプライベートリストを持つことにする。メタ配送リストを作成する際には他のサブリストのユーザリストと同様にプライベートリストも読み込むが、配送モード等はこれを優先する。

これにより、他のサブリストと独立して配送モード及び配送抑制の制御を行なうことが可能となる。

4 メタリストの実装

ARGO が用いる形式の配送リストを複数読み取り、メタ配送リストを生成するスクリプトを perl を用いて実装した。現在の実装で 316 行である。

メタ配送リストの生成には時間がかかるため、スクリプトは実行時に元となる各 ML のメンバリストと生成されたメタ配送リストのタイムスタンプを比較し、元メンバリストが変化していなければ新たなメタ配送リストを生成しない。

さらにコマンドラインオプションにより、メタ配送リスト生成時に、リスト生成とは別に標準入力からの入力を標準出力に転送する機能を持たせた。この転送はメタ配送リスト生成後に開始するため、メール配送スクリプトの前段にフィルターのように動作させることにより、メタ配送リストを更新してから、メール配送スクリプトを動作させることが可能になる。

5 メタリストの運用

筆者は ARGO を用いて「桜井智 ML」[5] および「長崎崩 ML」[6] の 2 つの ML を運用している。この 2 つの ML では共通の話題も多く、1 節で述べたような問題が実際に生じていた。そこで、実装したメタリストをこの 2 つの ML に対して適応し、2 つの ML のメタリストとして「エルスタッフ ML」[2] を 1997 年 5 月 15 日に作成した。

配送の形式的には新規の ML への登録となるため、メタリストが突然配送されると、各 ML のメンバにとっては容認していない新規の ML への登録が行なわれることになる。これは状況を把握していないユーザからの反感が予想されたため、メタリストのユーザ全てを一度配送停止状態でプライベートリストに登録し、各ユーザにコマンドメールによって配送を許可してもらう形式をとった。

メタリスト動作開始後のサブリストへの加入者には、無条件にメタリストの配送を行なった。新規加入者においてメタリストの配送を拒否したユーザは存在しなかった。

エルスタッフ ML は 1998 年 3 月現在、サブリストとして 394 アドレスの桜井智 ML と 168 アドレスの長崎崩 ML を持ち、さらに初期に登録した 287 アドレスのプライベートリストを持つ。BSD/OS を動作させている Pentium 100MHz の IBM-PC 互換機においてこの ML のメタ配送リストを生成するのに要する時間は約 3 分間である。

メール配送時のメタ配送リスト生成による遅延を避けるため、定期的 (1 時間毎) にメタ配送リストの生成スクリプトを動作させ、メタ配送リストを更新している。この際にもサブリストが更新されていなければ、メタ配送リストの更新も起こなわない。ML へのメールの到着は稀であり、メンバリストの更新はさらに稀であるので、実際の配送で遅延が問題になる事はほとんどなかった。

6 問題点

本節ではメタリストの運用からほぼ一年が経過した現在、明らかになった問題点を述べる。

[2] エルスタッフは二人の所属プロダクション名

表 1: メンバ管理関係のコマンド一覧

コマンド	操作内容	操作対象
member	メンバリストの取得	メタ配送リスト
subscribe	新規メンバの登録	プライベートリスト
unsubscribe	メンバの削除	プライベートリスト
mode	配送モードの変更	プライベートリスト

1 実装上の問題点

ARGO システムはメールアーカイブの検索やメンバリストの取得などをコマンドメールで遠隔実行することを可能にしている。メタリストにおいてもこの機能は有効であり、それ自身一つの ML として機能する。

しかしながら、プライベートリストという考えを導入したために、メンバ管理関係のコマンドを生成されたメタ配送リストとプライベートリストのどちらに反映させるかという問題が生じた。関係するコマンドとその現在の操作対象を表 1 に示す。

本来 1 つのメンバリストを管理するコマンド体系を無理に 2 つのメンバリストに適応しているために、サブリストにのみ登録されているユーザが取り寄せたメンバリストに記載されているからと unsubscribe コマンドを発行しても実行できないという混乱が生じた。

混乱を低減させるため、サブリストのみ登録されているために実行できないコマンドを発行した場合には、どのサブリストに登録されているかを返答するよう改良を行なった。また、プライベートリストを含めどのリストに登録されているかを返答するコマンド groups を新設し、ユーザがリストの状況を把握できるようにした。

しかし、メンバリストが 2 種類あるという現状は、一般のユーザにとってわかりやすいとはいえない。

2 運用上の問題点

メタリストの機構は非常に特殊で複雑な形態の ML であり、一般ユーザに理解されにくいという問題が生じた。特に、今回の試験運用を行なったエルスタッフ ML の場合、メタリストの利用方法として以下の 2 つを挙げた。

- 両方の ML に共通の話題
- どちらにも含まれないが、エルスタッフに関係する話題

このうち、前者はメタリストで扱うのに相応しいが、後者は受けとりたくないユーザが存在するため、区別すべきであった。

このこともあり、最初に配送停止状態とされているメタリストの配送を開始しないユーザが多数生じた。この結果、本来の目的の「全てのユーザへの通知」という機能が完全には機能しなくなってしまった。

7 今後の課題

ARGO システムは改良しやすく、高機能なものを目指して設計されたが、一方でコードおよびコマンド体系が肥大化、複雑化し、一般のユーザにとって使いにくいものになりつつある。

特に近年は計算機に関する知識を持たない一般の人々が道具としてメールを使う事が多くなり、容易に利用できるシステムが求められてきている。また、このような人々には計算機の世界の常識は通用しないため、ユーザの常識に頼らない強固さや柔軟さも必要である。

メタリスト機能は思想と異なる実装を行なったがゆえに ARGO システムに複雑さと歪みをもたらしている。試験運用を行なっている ML でも一年を経過してメタリストが認知されつつあるため、プライベートリストを扱わないシンプルなメタリスト機能を実装し、これに切り替えることによってメタリストとしての機能を果すようにすべきである。

また、メタリストの機構を平易に解説する文章が必要である。

8 おわりに

ML サーバ ARGO に、複数の ML のメンバーリストの和集合をメンバーリストとする、メタリストを作成する機能を追加することにより、共通の話題をこのメタリストで扱うことを可能にした。

今回実装したメタリストはその複雑さによって混乱を生じたものの、ML を流れるメッセージの一貫性の保持にある程度寄与し、その効果は確認できた。

今後はわかりやすい ML サーバシステムの構築をおこなっていく予定である。

参考文献

[1] 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, Apr., 1996.

[2] 楯岡 孝道,"自動応答機構 ARGO コマンド処理システム," 1996 年 調布祭 百万石, pp.7-11, MMA, Nov., 1996.

[3] 楯岡 孝道,"メーリングリストサーバ ARGO のダイジェスト配送," 1997 年 新入生歓迎シーズン 百万石, pp.2-9, MMA, Apr., 1997.

[4] 楯岡 孝道,"Hobby Softwares: ARGO," May, 1997.
http://delegate.uec.ac.jp:8081/club/mma/%7Etree/soft/#ARGO

[5] "桜井智 ML 「智の会」の紹介," 1995.
http://www.wp.com/STREAM/mli.html

[6] 楯岡 孝道, "Moe Observation Eminence," 1997.
http://delegate.uec.ac.jp:8081/club/mma/%7Etree/moe/ml.html

delegate越しにMMAにアクセスしてみよう計画

uirou@mma.club.uec.ac.jp

1998/03/25

1 なぜこんなことを考えたか(些細な必要性)

僕はWeb上で日記を書いている。[1] その日記はrsync[2] を用いてupdateを行っている。とりあえずは僕の入っているプロバイダさんに置かせてもらっている。案外プロバイダさんでも好きなことは出来るのだが、いかんせんMMAのように本当に好きなことが出来るわけではない。そこで、この日記をMMAの方[3] へ移動しようと考えた。

しかし、以下のような問題点が発生した。

- MMAは電通大内に存在していて、publicなIP addressは持っていない。このため、MMAに外部(例えばプロバイダ等)から直接rsyncすることが出来ない。

一応publicなIP addressを持っているマシンへ一度rsyncしてから、再度MMAにrsyncをすれば良いのだが、そんなまどろっこしいことをしたくは無い。

ところで、MMAはdelegateというproxy serverを利用して外部へ情報発信をしている。このdelegateというものは、いろいろなプロトコルを中継してlocal IPのマシンを外部から見えるようにしてくれる機能を持っているのだが、その機能の一部を利用してhttpだけは外部への接続が許されているわけだ。

http というものは、一応サーバとクライアントでデータの送受信が可能である。ということは、うまいことそのデータの送信と受信をしてやれば、MMAのデータを更新する程度であれば、案外簡単に作れるのではないだろうか。

2 実装方法(考えてみる)

今回の目的は、自分のWebPageのupdateであるのだから、localで作成したデータをtar.gzかなんかにして、POSTで送り付けてweb server側で展開してやれば良いように思えた。

しかし、この方法ではCGI scriptがhttpd権限で動くことや、ユーザーの認証などをしなければならない。それに、いちいち全てのデータを上書きしていたのでは、いつかは破綻してしまうのが目に見えている。やはり、必要なデータだけを送るようにrsyncのようなことをしなければならない。ということは、データの送信だけでは駄目で、サーバ側とのデータ送受信をするべきである。

はて、どうせサーバ側とデータの送受信が出来るのならば、それをsshd及び、rsyncへのpipeとして実装してしまってはどうだろう。そうすれば、ユーザ認証も必要無いし、httpd権限で動くこともない。もちろん、rsyncのやることをわざわざ実装する苦労からも解放される。

すばらしい。僕はpipeを作るだけで良いのだ。

3 仕様(仕組み)

さて、どのようにして実現するか、である。データの受信をするのであるから、受信側はPOSTを用いたCGI scriptが使われることは容易に想像できた。では、データの送信はどうであろうか。これは、nph scriptを用いることにより、server側から好きな時に好きなデータ量だけを送信することが出来るらしいのでこれを利用することにする。出来ればnph scriptそのものでPOSTも同時に受けたいところだが、POSTはContent-Length行の分だけhttpdがバッファしてしまって、接続を切られてしまう[4] ので、それは出来ない。ということで、web server側にはPOSTを受けるCGI scriptと、データを送信するnph scriptの二種類が必要になる。そこで、以下のような形で実現させることにした。

---

[1] 最近恥ずかしいことばかり書いているので場所は書かない。MMAに入れば教えてあげやう

[2] http://samba.anu.edu.au/cgi-bin/rsync/

[3] http://delegate.uec.ac.jp:8081/club/mma/

[4] Keep-Aliveを用いれば接続は切られないが、Content-Length行は必須なので結局それ以上のデータも送れないし、それ以下のデータも送れない。

```
 Server side                                  Client side
sshd <=> nph-script =======>+GET---------+
           |                 |  cilent   |
           |                 |  program  |<=== local port
POST を受け取る CGI script <= +POST--------+
```

というように、nph script、CGI script、そして local に置かれた client program を用いて local な INET domain な port を web server 側の sshd へと繋げてしまうのだ。

要するに local で client program を立ち上げておけば、

```
% ssh -p port localhost
```

とするだけで web server へ ssh で connect 出来てしまうわけである。

4 実装(作ってみる)

ということで、作ってみた。[5] 結論から言うと、動いた。速度は、、とんでもなく遅いが。実際、rsync でなく、ssh で login してみたが、きちんと term も制御されたりして、なかなか楽しい。X のプログラムも飛んで来るところなど、楽しくってしょうがない。

技術的な難しさはあまり無い。実際、nph でデータを吐き出して、POST でデータを受け取るところだけが出来れば、あとは簡単である。で、まだ高速化など、いろいろ手直しが出来そうなところは沢山あるので、そういうところをいじるほうが楽しそうである。

5 現状(バグ)

とりあえず動いてはいるのだが、問題点や疑問点がいくつかある。

- POST でデータを送る時に、何も考えずに "Content-Type: application/octet-stream\r\n" と書き、"Content-Length:" をきちんと送ってはいるものの、そう書いたからと言ってデータをそのまま何も加工せずに送ってしまっても良いのか？POST する場合は URL でないからエンコードは必要無い？これは本当？
- nph script は、sshd からの接続が切られた場合、UNIX domain socket からのクライアントのデータ入力が 0byte だった場合、select(2) で stdout が write 出来ないと判定された場合、のいずれかが成立すると exit する。が、どうも select で stdout を見張っても client が接続を切ったことはわからないらしい。かといって、client との接続は stdout しか無いわけで、、client が強引に落された場合、process が残ってしまうことがある。この場合、その process を殺すにはそのサーバに login して httpd 権限で殺すか、UNIX domain socket に NULL byte かなんかを入力するしか無い。
- 実験中に X の window を飛ばすと nph script が謎の exit をすることがあった。
- 実験中に delegate が ERROR: IMMATURE REQUEST BODY と言って POST が腐ることがあった。
- rsync だけがしたいのであれば、ssh proxy が上がっているだけで存在価値が無くなる。

これらの問題はもうちょっと調べればわかりそうなことだったり、気にしなければとりあえず OK な事だったりするので、これが製本される頃には直っているかもしれない。

6 将来(野望)

とりあえず現状ではかなりタコな実装になっている。例えば、一度に BUFSIZ だけしかデータを送らない、Keep-Alive せずにいちいち connect しなおす、等である。これらを改良すれば、もうちょっと高速化出来るものと思われる。

参考文献

RFC1945 Hypertext Transfer Protocol – HTTP/1.0
RFC2068 Hypertext Transfer Protocol – HTTP/1.1
WWW HTML memo (ftp://ftp.lab.kdd.co.jp/KDD/www-memo.tar.gz)

[5] select を使っているのに、fdopen したりするとかいう、お約束以前のミスをしたとかいう話は書かない

MMA FAQ Ver 1.0.0

uirou@mma.club.uec.ac.jp

1998/04/05

1 目次

これはMMA(Microcomputer Making Association)についてのFAQ(Frequency Answerd Quetions)をまとめた[1]もので、以下の項目が章毎に別れています。

- MMAの基本的なことについて
- MMAの年中行事について
- MMAの食事条件について
- MMAの人材関係について
- MMAの環境について
- MMAにおける学業について
- MMAの金銭面について
- その他

また、これらの質問は、初期において聞かれる質問に限定し、コンピュータの扱い方等の込み入った問題は取り上げないことにしています。

2 MMAの基本的なことについて

Q MMAって何ですか

Microcomputer Making Associationの略です。意味的には、「マイクロコンピュータを作る会」となります。といっても、最近はコンピュータを作ることはしていないですね。各自コンピュータを使って好きなことをしていると言うのが現状です。

Q MMAはどこにありますか

MMAの部室はサークル棟二階の奥にあります。途中でまに研に吸い込まれたりしないように注意しましょう。

Q MMAって普段何をしているんですか

難しい質問です。MMAでは個人で各々好きなことをしていて、MMAとしてまとまって何かをすると言うことは最近では行われていません。[2] 言い替えれば、自分だけが好きなことをするためにMMAの部費を使うことが出来るかもしれない、ということになります。あと、別に複数人で何かをするということが出来ないと言うことも無いでしょう。あなたが何かをしたくて、それが他人の協力を得るべきな物であれば、MMA部員に助けを求めるのは良い考えです。

Q MMAってお金がかかりますか

少しだけかかります。入部金が3000円で、その後月に1000円の部費が徴収されることになっています。ただ、実際には部会のときなどにしか部費を集めていないので、部会のときに「あ、ごめん。今お金無いんだ」とか言いつつ、逃げることが出来れば、かなりのあいだ払わないでもなんとかなっちゃったりするようです。過去においては、実際に逃げ続けて二年間以上部費を請求されないですみそうになった人の事例も報告されています。[3]

Q MMAには何曜日に出て来るべきなのですか

これといって決まった曜日の指定はありません。というか、皆好きなときに部に出て来ているだけだったりします。統計的にみると、昼休み、四限の授業が終った後、あたりにはアクティブなメンバーは部に顔を出しており、大体夕飯を食べに行ったりしない限りは21時から22時あたりまで部室に居るようです。また、Network的に部にloginしている人も最近では増えており、部に実体は居なくても、phoneやIRCを使ってお話が出来たりします。

[1] でっちあげたとも言う
[2] ただ、最近MMAらしい内容のセミナーを隔月程度の頻度で行おうという動きがあります
[3] でも、最終的には払わなければいけないのは理解しておいてね

Q いつ部員が来るかわからないのに、どうやって連絡を取っているのですか

MMA 現役メーリングリストというものがあります。電通大に在籍している人は少なくとも CC[4] にはアカウントがもらえますし、最近の MMA 部員は何らかの方法で自宅やバイト先からメールが読める環境にいるようです。そのメールアドレスを利用して普段顔を出さない人でも密に連絡を取り合っています。まぁ、はじめのうちは部に顔を出さないと顔を覚えてもらえませんので、こまめに顔を出すようにしてくれたほうがこちらとしてはありがたいです。

Q MMA にはどのくらいの部員がいますか

だいたい 25〜30 人位の部員がいるようです。そうすると、部費だけで年間 300k〜360k ものお金が入るはずなんですね。すごいなぁ。

Q MMA に入るときは、どんなことをすれば良いのですか

僕／私は MMA に入りたい、という意志表明をして、入部金を払うだけです。基本的に入部金を払った時点で MMA 部員として認められ、MMA でのアカウントが掘られます。

Q 入部金を払わないと部に顔を出してはいけないのですか

そんなことはありません。体験入部という形[5] であれば、別に入部金や部費を払わずとも guest アカウントを用いて遊ぶことが出来ます。ただし、鍵システムの関係上、guest アカウントでは部に入ることが出来ないので、部室に自由に出入りしたい場合は、部員になるべきなわけです。

3 MMA の年中行事について

Q 合宿ってありますか

あります。毎年、夏に一回、広い場所が近くにあるところへ遊びに行きます。

Q 合宿ではどんなことをするのですか

基本的には OB のために[6] 花火を上げるだけです。と言っても普通の打ち上げ花火ではなく、「煙火打ち上げ従事者」とかいう免許がいる程度の花火[7] を上げます。これは、n 号玉と呼ばれる直径 5cm〜8cm 程度の火薬の塊を打ち上げると言う、下手をすれば腕が吹き飛んじゃったりすると言うぢつはかなりとんでもないものだったりします。で、花火を上げる以外は何をしているかと言うと、日頃のコンピュータと密に接触している生活から離れ、のんびり、ぐーたら、することになっているようです。[8] そのため、8bit 以上の CPU を搭載した物は持ち込んではいけないことになっているはずです。しかし、最近はノートパソコンが普及してきていて、それを偉い人々が所持しており、なに喰わぬ顔で合宿に持ち込んだりしているので、最近では原則が破られつつあるような感じです。[9]

Q 僕／私も煙火打ち上げ従事者になれますか

なれます。たった一万円を払って一日だけ講習会に出るだけであなたも立派な煙火打ち上げ従事者です。もしも希望者が少なければ、その一万円ですら MMA が負担して講習会に出させてあげることが出来ます。

Q 合宿のほかにはどんな行事がありますか

合宿の次に大きなイベントは学園祭です。それ以外のイベントは、そう。この時期の新入生歓迎、略して新歓です。MMA としても、新たなる金づる、、あいや、期待の新人を獲得するための重要な行事です。

Q 学園祭ではどんなことをするのですか

一昨年当たりの学園祭からは MMA においてぢゃんく屋[10] が開かれ、かなりの好評を得ています。ですが、学園祭における一番のイベントは OB[11] 参加の「鍋」で、これも MMA 現役部員と OB との接点となっています。

Q MMA ではジャンク屋をやると聞きましたが、本当ですか

---

[4] 電通大の学生全員が使えるコンピュータセンター
[5] そう呼んだほうが何かと都合が良いだけです
[6] そう。主役は OB なのである
[7] 煙火というらしい
[8] もちろん例外はありますが
[9] 筆者も、ノートマシンを買ったら持って行きたいなぁとか思いそうです
[10] 「ジャンク屋」とは書かない
[11] やっぱり OB は大切にしなければならない

半分本当です。というのも、ぢゃんく屋をやるのはMMAという団体ではなく、部員の一人がMMAのブースを借りてやっているという形を取っているからです。もちろん、部員がやっているのですし、ブースはMMAの中にあるのですから、MMAがやっているのとおんなじような気がするのも間違いではないでしょう。ですが、実際に売子をやらなければならなかったりする強制力が働かなかいようなところが微妙に違うわけです。

Q 新勧ではどんなことをするのですか

基本的に百萬石[12]を配布したり、尋ねて来た新入生に生のMMAをみてもらったりします。そして、4月の終り当たりに新入生歓迎のコンパを開きます。これは、毎年深夜遅くまでかかるらしく、大抵の人物が泊まってから始発以降の電車で帰るため、だいたい金曜日に行われるらしいです。[13]

4 MMAの食事条件について

Q MMAでは普通どんなところに食事に行くのですか

筆者は調布に住んでいないのであまり良くは知らないのですが、とにかく量が多いまんぼう、ご飯お代わり自由のさぼてん、学生であれば大盛り料金ロハのそらまめ、等が近くて量があっておいしい所のようです。

Q まんぼうって何ですか

CURRY HOUSE MANBOWという、カレー屋さんのことです。MMAの中では一番人気の高いところで、店のおばちゃんに顔を覚えられている部員が多数おり、忘れ物をしたときには次回来たときにきちんと忘れずに渡してくれる程、親しい仲になっているようです。基本的には普通のカレーなのですが、とにかくご飯の量が多く、大食漢のあなたでも満足させることが出来るでしょう。また、セットメニューを頼むと、ご飯の他にナン、サラダ、ヨーグルト、ジュースが付きます。ナンはご飯の代わりみたいなものなので、これの大盛り、ともなれば、日に1回しか食事をしないような不規則な生活のときにでも十分に対応できる程度の量[14]を食べることが出来ます。また、この御店には激辛カレーというものが存在します。筆者はまだ食べたことは無いのですが、その辛さと言ったら、次の日のトイレが辛いことになるほどだそうです。そのため、MMAでは伝統的に激辛カレーを食べる場合はおごってくれると言うものがあるようです。しかし、もしあなたが激辛カレーに自信があるとしても、MMA部員の前で「僕、まんぼうの激辛に挑戦してみようかなー」等とは言うべきではないでしょう。まぁ、中には激辛を普通に食べられる人もいるので、無理に挑戦するな、とは言いませんが、、

Q さぼてんって何ですか

さぼてんという豚カツのチェーン店のことです。御値段は980円〜1400円程度と、けっこう張りますが、豚カツ以外(ご飯、御味噌汁、サラダ)はお代わり自由と言う、非常にありがたいお店です。昔はテーブルに振りかけが用意されていたのですが、最近では無くなってしまったようです。さすがにご飯だけ大量に食べられるのは嫌だったのでしょうか。

Q そらまめって何ですか

そらまめ拉麺本舗というラーメン屋さんの事です。ここでは、学生証をみせると大盛り料金がタダになるというサービスを行っており、お腹を空かせたMMA部員は当然そのサービスを利用するわけです。味のほうもそれなりに美味しいです。

Q MMAに入って激辛カレーを食べると偉くなれると聞きましたが本当ですか

多分嘘です。と言っても、まんぼうの激辛カレーを食べ切った人は、大抵偉い人になっています。実際、偉い人で激辛カレーを食べていない人と言うのは、数える程しかいません。きっとハッカーと激辛と言うのは、何か数学的な関連性があるのでしょうね。

5 MMAの人材関係について

Q MMAにいじめはありますか

---

[12] 今、読んでいるこれのことね
[13] ということで、自宅生はその日のうちには帰らなくても良いようにきちんと裏工作しておいてね
[14] そのくらいお腹を空かしていないと食べきれないのでは？

あるという人もいれば、無いと言う人もいます。というのも、MMA はそれなりに人数が少ない部なので、けっこうなぁなぁな雰囲気でそれぞれがちょっかいを出し合っているようです。まぁ、矢印は存在するようですが。それはそれぞれの性格によるのでしょう。

Q MMA に恐い人はいますか

います。いろんな意味で恐い人がいます。御飯をおごってもらうと高くつく人、論戦をふっかけるととても楽しそうに応対してくれる人、借りを作るといつまでも付きまとう人、いろいろいます。でもまぁ、そのうちなれて、上手に立ち回れるようになるでしょう。

Q MMA で本名を使っていない人がいるのですが、これはどういうことですか

MMA では伝統的に alias[15] をアカウントに使用しています。この alias はこれまた伝統的に先輩達がほとんど勝手に決定してくれちゃったりします。この alias というものは、かなり重要な位置を占める場合があり、最悪、MMA を卒業しても、本名よりも alias の方が有名、もしくは知らない人から alias で呼ばれる、等の事態が発生し得ると言うような感じです。ちなみに、alias には変なものが好まれるようです。[16]

Q MMA はヲタクなんですか

ヲタク、という言葉を筆者はいまいち理解していないのかもしれませんが、MMA はコンピュータを主に扱う部であるがゆえに、部員がモニタに向かって黙々とキーを叩き続けるという症状がしばしば見受けられることになります。[17] そういう点から行くと MMA はヲタクなのかもしれません。

Q 猿ってなんですか

語源はよくわからないのですが、一見価値の無いこと[18] を繰り返し繰り返しやり続ける状態のことを言います。猿状態の人物は大抵は価値の無いことをしているので、猿、というのは差別用語なのではないでしょうか。

Q MMA に女の子はいますか

今のところ、いません。だれか良い娘[19] をつれて来て／入って下さい。そうすれば、少しは MMA の駄目な雰囲気も一掃されるでしょう。

Q MMA に入ると企業の人とお知り合いになれるって本当ですか

本当です。MMA の OB はかなりその手の分野での偉い人が多く、合宿や学園祭などで OB の方から尋ねて来てくれたりするので、そのときに接点が出来るようです。まぁ、電通大という学校自体、その手の分野の偉い人をいっぱい輩出しているわけですけどね。

6 MMA の環境について

Q MMA にはどんなものがあるのですか

MMA のメインの機材は基本的にコンピュータと呼ばれるものになります。そのほかには各種技術書、各種工具、各種測定機器、過去問、過去レポ、ガスコンロ、鍋、バーベキューセット、各種趣味に合いそうな本[20] 、なんだかわからないもの、等があります。あなたは MMA に入ることによってこれらの物を使う権利を得ます。

Q MMA にはどんなマシンがあるのですか

最近の目玉は alpha マシンです。これは、学校側がサークルに対して使うお金を MMA に少し回してくれたおかげで手に入ったもので、執筆時点の MMA では文句無しに一番速いマシンです。Pentium なんかより浮動小数点演算が速かったりするので、筆者はちょっといろいろ楽しいことが思いついちゃってにこにこしちゃいます。で、それ以外のマシンは、IBM PC-AT 互換機、SUN マシン、などがあるようです。基本的には IBM PC-AT 互換機を組み立ててメインのマシンにしたり、学校から投棄された SUN マシンなどをどこからか get してきて再生させて使っていたりします。もちろん、あなたが使いたいアーキテクチャのマシンを持ち込んでも問題無いでしょう。

Q 部に顔を出している間はメールが読めるって本当ですか

---

[15] ニックネーム。あだなとも言う
[16] この意味はそのうちわかるようになるでしょう
[17] 今、これを書いてるときだってそうだし
[18] 実際価値の無いことだったりする
[19] 何かと世話好きで、面倒見が良くって、仔犬チックな、、、すみません。殴らないで下さい
[20] 定期的に補充している人がいるらしい

本当です。MMA は3年前から電通大内の LAN に接続されており、MMA へのメールの配送も出来るようにしてもらっています。もちろん、Jed、Sed、Med[21] 、CC 等に MMA から login することも出来るので、そちらが混んでいたりしそうで行きたくない、とか、いちいち歩いて行くよりも MMA の方が近いから楽、なんて時には便利だったりします。

Q MMA には NEC-PC98 シリーズを持ち込んじゃいけないと聞きましたが、本当でしょうか

昔はそういうお話もあったようですが、今では別に構いやしないのではないでしょうか。というか、筆者はなぜ NEC-PC98 シリーズが部室に出入り禁止を食らったのかしりません。というか、その位昔のお話のようなので、もう時効でしょう。

Q MMA ではどんな OS を使っていますか

基本的には UNIX 互換の OS を使っているようです。だからといって、BeOS は使わない、とか、Plan9 は使わない、なんてことはありません。やりたければ「僕、○○ OS で遊んでみたいー」と言い出して、マシンを一つ get すれば良いのです。ただ、MMA も現在では電通大内部のネットワークに接続されている状態なので、外部へのサービスや、メールの受信などのために、サーバを上げておかなければいけないので、それを現在の UNIX から他の OS に変えるのはちょっと大変でしょう。

Q MMA では MS-Windows は扱わないと言うのは本当ですか

今のところ本当なんじゃないでしょうか。でも、永久に Windows が部室にはびこらないとは言えないと思います。[22] 例えば、現在これを執筆中のマシンでは、MS-Windows95 の上の ASTEC-X を用いていたりするのです。多分、Windows なソフトを書く[23] 人材がいない事や、Windows ではマトモに遊べるソフトが少ない[24] からではないでしょうか。MMA は、そもそもゲームをするための部じゃないですし。でも、もしもあなたが MS-Windows を使って、何か楽しいことをしたいと言うのであれば、問題ありません。部会などで発言し、うまく部員を誘導することによってマシンを Windows 用に一つ確保することが出来るでしょう。ただ、部室のマシンを全て Windows に変えようとすると、相当な努力が必要になりそうな気がします。

7 MMA における学業について

Q MMA には過去問、過去レポはありますか

あります。が、完備しているとは言いがたい状況です。やはり、部員の数がそれほど多くないので、必然的に過去問、過去レポの数は少なくなります。得に、J 科以外の学科になると、部員の存在確立が減るので、もっと数が少なくなる可能性があります。ただ、一年生の場合は、どの学科でも同じような授業をうけますし、だれしも一年生の時は真面目ですので、物理／化学実験などのレポートは結構量があるようです。[25]

Q 単位はちゃんと取れますか

取れます。それは、単にあなたの学業に対する気持ちにかかっています。MMA に入っていて、アクティブに活動していて、きちんと単位を取っている先輩はいます。

Q MMA に入ると人生変わりますか

変わる人もいれば、変わらない人もいるようで、いまいちどちらとも言いかねます。ただ、変わる場合はいろいろなパターンがありそうです。たとえば、MMA に入って、布教されて、春休みを全てそれに捧げてしまうパターン[26] 、MMA に入ってバイトにいそしんで、そのまま学校にあまり顔を出さなくなってしまうパターン、MMA に入ってバイトにいそしんで、そのまま就職してしまうパターン、等です。しかし、MMA にいてもきちんと学業の事を考えて怪しげなパターンにはまらずに[27] 順調に単位を取り、卒業する先輩もいます。ということで、やっぱりあなた次第です。[28]

Q MMA に入ると卒業できないって聞きましたが本当ですか

---

[21] J,S,M 科のコンピュータセンター
[22] MS-Windows が永久に存続するとも思えませんが
[23] 書けるのと書くのは違います
[24] ゲームは除く
[25] 内容は保証しませんが
[26] 目線が僕に集中しているのは気のせいに違いない
[27] いや、たとえはまったとしても
[28] 強い意志と行動力があれば、、

嘘です。卒業できないって事はありません。実際に毎年追い出しコンパはきちんと開かれます。ただ、留年してしまう人がいるかもしれないということです。やっぱり、あなたの心がけ次第なわけです。

8 MMAの金銭面について

Q MMA に入るとバイト先を斡旋してくれると聞きましたが本当ですか

嘘です。バイト先を斡旋はしていません。単純に「使えるバイトを求めている会社」にバイトに行っている人が多いだけです。というのも、MMA に入っているような人は、総じてコンピュータについていろいろ知りたい、とか、コンピュータをいじっていて楽しい、と言うような人ですので、自ずとそっち方面の会社では入った途端に「使えるバイト」として活動することが出来るような感じがするようです。[29] そのため、その手のバイトに行っている人の前で、「僕も何かバイトしてお金を稼ぎたいなぁ」とか、「最近暇だなぁ」などと言うと、「良いバイト先があるよ」という話を持ち出されてしまうわけです。

Q MMA に入るとお金を使うことが多くなりますか

MMA に入ったからと言ってお金を使わなければいけないと言うことはありません。ですが、今まで知らなかった物を知るために、本を買いたいと思うことはあるかもしれません。そして、そのような本は大抵技術書ですので、結構値段が張ります。でも、MMA 部室にはそれなりの本が常備されていたり、部員という生き字引がいるので、最初のうちは本など買わずに部室に顔を出すというのも良いでしょう。

Q MMA でバイトを斡旋すると報償金が出るって本当ですか

初耳です。報償金が出るのなら、僕にも下さい。でも、バイトを紹介してくれた人に、感謝の気持ちを込めて食事をおごったりするのは良いことでしょう。

9 その他

Q MMA の部室に入るときに、キーボードを叩いていますが、あれは何ですか

あれは MMA の鍵を管理している鍵サーバへ自分のアカウントでもって login しているのです。実際は部員が login すると鍵が開き、閉めるためのアカウントで login すると、鍵が閉まるようになっています。なぜこのようなシステムにしたかと言うと、昔、部室の鍵が他の部と同じような鍵だったとき、部室で盗難事件が発生したからです。当時かなりの高級な材料を使ったマシンが一夜にして盗まれてしまったときは部員がみんながっかりしたものでした。そして、学生課のずさんな鍵の管理体制のため、犯人の特定は非常に難しく、泣き寝入りをする程度しか選択肢はなかったのです。しかし、鍵サーバが実現されてからはそのような盗難事件は発生せず、平穏な部室が戻って来ました。もしあなたが鍵サーバにアカウントが欲しい場合は、部員になってから要求すれば、鍵サーバにアカウントが作ってもらえます。[30]

Q MMA に入るとあっちの世界が見えてくると聞きましたが本当ですか

「あっちの世界」というものがどういう世界かは良くわかりませんが、筆者は、コンピュータ業界や、ゲーム業界、二次元女の子関連、などの「あっちの世界」は、ちょっとだけかいま見た気がします。といっても、みたからどうだということは、、、無いんじゃないかなーと思います。[31]

Q 布教って何ですか

自分の信ずるものや事柄を他人に広めようとする試みのことです。通常は宗教について当てはまるものなのですが、MMA においては宗教家は存在しないのでそのほかの「その人の信ずるものや事柄」を、まだ信じていない人に広めようとするわけです。まぁ、自分の趣味を人にも教えて、同じ趣味の人を増やそうという試みなわけですね。

Q いままでで一番嫌だったことは何ですか

---

[29] 実際使えるかどうかは、それぞれのスキルや性格にもよると思いますが

[30] ただ、きちんと要求しておかないと、作るのが面倒とか、忘れた、と言う理由で作られない可能性があるようです。注意して下さい

[31] そこ！嘘だとか言うな！

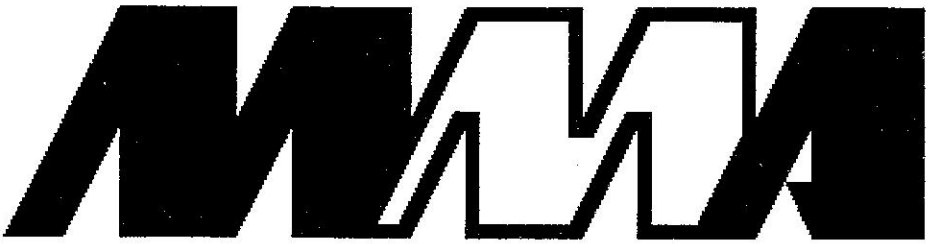
そうですねぇ、嫌だなぁ、と思ったことは過去にはありますが、それも楽しい思い出ととなってしまったので、どうということはないですね。最近、布教されたときには一瞬「嫌だなぁ」と思ったものですが、すぐに補完されてしまったようなので今は幸せです。[32]

Q MMA は宗教とかやっていますか

やっていません。ですが、毎年 MMA 宗教法人化計画なるものが発動しかけたりします。宗教法人は税金がかからないそうで、いろいろと楽しいことが出来るらしいです。税金のかからない状態で OCN とかで常時接続みたいなことをやるのでしょうか？それとも税金のかからない状態で不可思議なご利益の物品を高値で売り捌いたりするのでしょうか？ちょっと楽しそうです。

Q MMA のロゴってどうやったら描けますか

実は、案外簡単です。実際はこんな感じ



ですが、よくよく計ってみるとわかるように、最初の M の字の出っぱり部分を一単位とすると、高さがその四倍、斜めの線は縦横比が 2 : 1 である、とかそんな感じで比率が決まっています。一度計ってみて下さい。

[32] 錯乱しているようなので、気にしないで下さい

kou復活計画

fujita

概要

MMA には数台の PC の UNIX 端末がある。これはその中の一台、kou という hostname の PC の復活の軌跡である。

1997年11月某日その1 調布祭準備期間

調布祭のために MMA 部室内の PC(mola 以外) を西地区に運び出す。

1997年11月某日その2 調布祭第?日目

先日持ってきた PC 群を Network につなげることになる。OS から何から全て洗いざらい入れ替えねばならない。新規に入れる OS は OpenBSD に決定。インストーラが腐ってるのにもめげず、夜を徹して作業に励む。

1997年11月某日その3 調布祭第??日目

もう明け方だ。他の人達は眠っている。どうにか X-Window を入れるまでに至る。ところが X の設定でコケる。XF86Setup でマウスの設定が終わる前にマウスを動かしてしまったのが原因らしい。徹夜したから疲れてたんだろうか? とりあえずその状態で放っておく。どうせ部室に戻したらまた一から OS を入れなければならないのだから問題ない。

1997年11月某日その4 調布祭の後片づけ

PC を元あった部室に戻す。今後、気が向いた誰かが復旧作業にとりかかるのだろう。

1998年1月某日 新年会の帰り道にて

MMA の新年会である。仙川のピザ屋と調布の飲み屋をハシゴして、調布駅で解散。自転車を停めてある北口に向かう。e さんも大学構内に自転車があるため一緒に北口に出る。その道すがら e さんが、
「kou に FreeBSD 入れてみない?」
と、唐突に提案。調布祭以後、この時までに MMA の PC は元通りに戻っていた。しかしただ一台、kou だけが腐っていた。kou には e さんが OpenBSD を入れていたのだが、キーボードから入力が出来ないというドラスティックな症状が発生したため、e さんは kou のメンテナンスをあきらめて、最近購入したノート PC をいじくることにしたらしい。すかさず私が言い返す。
「いや、でもテスト期間中ですし...」
実際、次の週から後期テストが山のように予定されている。他のことに気を取られて単位を落としたらシャレにもならない。それに対する e さんの言。
「じゃぁテスト終わったらやってね。」
ちなみに調布駅で別れた他の人達は、その後 t さんの家に行ったらしい。そこで u さんが t さんに世界の果てを見せられたのは周知の事実である。
(http://delefgate.uec.ac.jp: 8081/club/mma/ sanjigen/diary/199801.html#199801)

1998年2月21日 e さんからのメール

メールを読もうとして mola に login。mnews で mma-active のメールや友人からのメールを読んでいると、その中の一つに e さんからのメールが。

```
Subject: FreeBSD install kou
```

どうやら e さんはこの前のことを忘れてなかったらしい。

1998 年 2 月 23 日 FreeBSD-2.2.5-RELEASE を入れよう

メールまでもらって何もしないわけにはいかない。ということで部室に行って netscape を立ちあげて http://www.jp.FreeBSD.org なんかをつらつらと読んでみる。夕方、他の人達はそれぞれ家に帰ったりお勤めに行ったりして、部室には私と e さんだけが残る。怒涛のインストールの始まりである。まず ftp://ftp.ee.uec.ac.jp/pub/FreeBSD/2.2.5-RELEASE/foppies から boot.flp を持ってくる。
で、dd if=boot.flp of=/dev/rfd0a でブートフロッピーを作る。そのフロッピーを kou の FDD に差して電源 ON! しばらくすると画面に Boot:と出る。ここですかさず e さんが UserConfig で立ちあげるように指示。-c と打てば良いらしい。どれどれ。おぉ、謎のメニューが。再びここで登場する e さん。真ん中の項目を選んで次の画面へ。HDD やマウス、FDD 以外の不要なデバイスを削っていく。ついでに Ether カード周辺の数値を変えて作業終了。ここから本格的なインストールの開始である。

インストール・プログラムの初期画面。Usage とか Doc なんぞには目もくれず Custom インストールへ。上の項目から順に設定していく。

まずは Options。ここは何もいじくる所は無いのでそのままにしておく。

次に Partition でハードディスクの設定をする。ディスクをまるごと使用するので A(Use Entire Disk) を入力し、続いて S(Set Bootable) を入力する。これでブート可能になったわけだ。Q(Finish) を打って終わろうとするとブートマネージャを使うか、などと訊かれるがそんなモノは使わないので Standard(no boot manager) を選んで Partition メニューを抜け出る。

三番目は Label である。ここで/、swap、/var、/usr の容量を決める。C(Create) を入力して手動でやる。まず最初にサイズを訊かれるのでメガ単位で何十 M という風に教えてやる。次にファイルシステムかスワップかを訊いてくる。/、/var、/usr の場合はファイルシステムを選んで次に出るマウントポイントの項目にそれぞれ/、/var、/usr と書いてやる。swap の場合はスワップを選んでやると OK。後、慣習として swap はメモリの 2 倍の容量にする。全部の設定が終わったら Q(Finish) を入力して終了である。

四番目の Distributions でインストールするソフトを選ぶ。この辺は適当なモノを詰め込んでおけば何とかなる。そんなもんだ。

次に Media でインストール元を決める。手っ取り早く FTP インストールを行う。なんせ FTP サイトが同じ敷地内にあるのだ。これを使わない手はない。ということで FTP サイトは ftp.ee.uec.ac.jp にしてインストール開始だ。

五番目の Commit を選べばインストールが始まる。カリカリ言いながらハードディスクが回転してる。もう窓の外は真っ暗だ。

ところが途中で/usr が書けないぜとか文句を言われる。やはりハードディスクの全容量が 8G とか非常識的なことを言ってた時点で変だったのだろうか?仕方ないので作業は一からやり直しだ。BIOS をいじったり Partition メニューでシリンダやヘッダ、セクタの数値を変えたりして、どうにかハードディスクの容量が正常値 1G に修正。さっきと同じ手順を踏んで再インストール開始。さぁ、今度こそうまくいくのか? ... 無事終了!良かった良かった。

引き続き設定をしていく。まず Add User。これは後で home を NFS マウントするので今のところ不必要だ。次に Root Password で root のパスワードを決める。さくさくと。Time Zone は時間の設定である。時間と分を指定して、タイムゾーンは Asia の Japan を選ぶ。この辺の設定はさっきまでの作業と比べて楽なもんだ。

Networking で

```
host name(kou.mma.club.uec.ac.jp.private)
Domain name(mma.club.uec.ac.jp.private)
Gateway(172.21.139.1)
Name server(172.21.139.43)
IP Address (172.21.139.36)
Netmask(255.255.255.0)
```

を設定する。

ここでメニューを終了する。マシンがリブートして login:というプロンプトが出る。FreeBSD のインストールは成功したようだ。/stand/sysinstall で再びインストールメニューを立ちあげて Configuration、Packages、FTP で各種ソフトの FTP インストールをする。FTP 元は前出の`ftp://ftp.ee.uec.ac.jp/pub/FreeBSD`。tcsh やら kterm やら mule やら canna やら mnews やら VFlib やら色んなものを取ってくる。ネットワーク的に近いとこういう時に幸せだ。

次に X の設定をするため/usr/X11R6/bin/XF86Setup を立ちあげる。まずは Mouse の設定をする。プロトコルは Microsoft、デバイスは/dev/ttyd0、Emulate3Bottons を指定して Apply する。マウスを

試しにぐりぐり動かしてみると、ちゃんとカーソルが反応する。よしよし。Keyboard の設定はデフォルトの英語 101 キーボードのままで OK である。Card は Mach32、Monitor の Horizontal は 31.5-64.3、Vertical は 50-90、そして解像度は 1024x768 を指定する。Other は特に何も変更するところは無い。Done を選択して/etc/XF86Config に書き出してセットアップ終了である。

さぁ、X はちゃんと動くんだろうか?行けぇ、startx!... あ、落ちた。どうも Monitor の設定をしくじったらしい。もう一回 XF86Setup を立ちあげて設定し直して... よし、再度startx! X なんてファイルはここにはねーぜ、とか言われて X の起動は失敗。どうも/usr/X11R6/bin/XF86_Mach32 を/usr/X11R6/bin/X にシンボリックリンクすればいいらしい。私がリンクの張り方を知らないので失笑する e さん。ln -s /usr/X11R6/bin/XF86_Mach 32 /usr/X11R6/bin/X とやればいいと教えてくれる。三度目の正直の startx!おっしゃ、成功。

いよいよインストール作業も大詰め、home の NFS マウントだ。mount nfs mola:/export/home /home でマウントする。これで各部員の home が kou からでも使用可能になったわけだ。ただ、これだけでは login できないので、ほかの端末から/etc/master.passwd を持ってきて kou の master.passwd にくっつける。さらにパスワードの暗号化のために DES をどこぞから持ってきてインストール。リブートするときのために/etc/fstab に 172.21.139.43:/export/home /home nfs rw 0 0 と書き加える。

それから su で root になるために/etc/group の wheel:*:0:と staff:*:20:の項目に私と e さんのアカウントを追加する。準備万端、logout して自分のアカウントで login してみる。あれぇ?入れないぞ?/bin/tcsh がないとかほざいてる。インストールしたはずなのに... tcsh のありかが分からなくておろおろする私。e さんが見かねて/usr/local/bin にあるからリンクを張ればいい、と教えてくれる。ln -s /usr/local/bin/tcsh /bin/tcsh でリンクを張って、再度自分のアカウントで login する。おぉ、入れた。

後は xdm が立ち上がるようにしてリブートすればインストール完成である。/etc/rc.local に

```
if [ -f /usr/X11R6/bin/xdm ]; then
        echo -n "xdm " ; /usr/X11R6/bin/xdm
fi
```

と書き加えて shutdown -r now、リターン! ... おっしゃ、xdm 起動。早速 login。あれ?入れない。e さんは入れるのに。.xsession が変なのかも、と e さん。直そうと思っても既に夜十時。学校から締め出される前に帰らねばならない。まぁ、ここまで来たら後の作業は大したこと無い。ということで帰ることにする。「lynx とか netscape とか入れてね。カーネルの再構築も。後、X をベクトルフォントにするとか。」別れ際に e さんがそう言った。私が kou の環境整備を終えるのはまだまだ先になりそうだ。

1998 年 2 月 27 日 lynx-2.7.1ac-0.84 を入れよう

lynx は文字ベースの WWW ブラウザだ。意外と MMA 内での利用頻度は高いと思う。ところが /stand/sysinstall を使って ftp://ftp.ee.uec.ac.jp/pub/FreeBSD/ を覗くとそこには無い。そこで色々と検索した結果、ftp://ring.aist.go.jp/pub/FreeBSD/packages-2.2.5/All にあったのでそこから持ってくる。package のインストールは簡単だ。/usr/sbin/pkg_add lynx-2.7.1ac-0.84.tar.gz で完了である。さて、ちゃんと機能するかどうか、試しに立ちあげてみる。おや、日本語が表示されないぞ?しかも UEC の外に繋がらないし。何でだ? ... え?皆マンボウに行くの?じゃぁ私も。

1998 年 3 月 2 日 UNIX MAGAZINE のバックナンバー

lynx の修理に何の打開策もないまま一日が過ぎる。先日部室の掃除の時にもらった UNIX MAGAZINE'95 年 3 月〜'98 年 2 月までを自転車のカゴにつめこんで家に帰る。特にやることもないので持ってきた UNI-MAGA を読んでいると、lynx を紹介している記事を発見。それによると lynx は日本語パッチを当てないと日本語を表示することが出来ないらしい。なるほど。これは良いことを聞いた。

1998 年 3 月 3 日 lynx 日本語化計画

早速http://www.goo.ne.jp で日本語パッチを探す。十数件ほどヒット。つらつらと読んでみると、日本語パッチはlynx-1.?.?用のものでlynx-2.7.1 は単独で日本語が表示できるらしい。ますます混乱して端末の前でもんどりうってると、-cfg オプションで config ファイルが指定できて、デフォルトで/usr/local/etc/lynx.cfg を読みに行くことを発見。もしかしてこのファイルに何かあるのか?と思って中身を覗いてみる。それら

しいところはあるんだけど、下手に手を出したら一層ヤバいことになりかねないので別の方法を考える。そう言えば他の端末では lynx がちゃんと動いてる。ということは他の端末の lynx.cfg を参照すれば良いのではなかろうか。ということで config ファイルの間違い探しを始める。案の定 CHARACTER_SET と proxy の設定が違ってる。早速修正。デフォルトの CHARACTER_SET:ISO Latin 1 は#でコメントアウトして CHARACTER_SET:Japanese (EUC) に変更。

```
http_proxy:http://delegate.uec.ac.jp:8080/,
ftp_proxy:http://delegate.uec.ac.jp:8080/,
no_proxy:uec.ac.jp
```

の三行を追加する。さて、これでうまくいくんだろうか。lynx を立ち上げてみる。おやまあ、ちゃんと日本語が表示されてるじゃないの。UEC の外にもつながるし。偉い偉い。苦労した甲斐があったというものだ。

1998 年 3 月 4 日 netscape-3.04 を入れよう

調子にのって netscape のインストールを試みる。http://home.netscape.com/ja/ を見てみるが何の情報も得られない。package になってないかなぁと思って ftp://ftp.ee.uec.ac.jp/pub/FreeBSD/packages-2.2.5/www/ を覗いたりしたが何もない。当り前だが。package でなければ ports がある。そう思って今度は/pub/FreeBSD/ports-2.2.5/www/ を見てみる。するとそこには netscape-3.03 と 4.03b8 の ports があった。あとは/pub/FreeBSD/distfiles/に netscape のバイナリがあれば言いわけだが、バージョン 3.04 しかない。3.04 の ports を求めてさまよってると/pub/FreeBSD/ports-3.0/www/ で発見。ports のインストールにはファイルからディレクトリまでそっくりそのまま落してこなければいけないらしいので、ちょっと疲れる。/pub/FreeBSD/distfiles/netscape-v304-export.x86-unknown-bsd.tar.gz は/usr/ports/distfiles/に、/pub/FreeBSD/ports-3.0/www/netscape-3.0/以下のファイルとディレクトリ群は/usr/ports/www/netscape3/に落とす。su で root になり、/usr/ports/www/netscape3/に移動して make する。無事終了したので make install。終ったら注意書きが出て来たのでその通りにする。

```
cd /usr/X11R6/lib/X11/fonts/misc
/usr/X11R6/bin/mkfontdir
chmod 444 fonts.dir
```

試しに立ち上げてみる。netscape、リターン。netscape:Command not found...? /usr/local/bin/には netscapeってファイルがあるのに。何でだ? http://www.jp.freebsd.org/ ryuchi/QandA/HTML/231.html によると http://www.bpel.tutics.tut.ac.jp/ take/Netscape/ を読むと何かわかるらしい。読んでやろうじゃないの。ええと...netscape も lynx 同様日本語対応にしてやる必要があるのね。はぁ...(ため息) 困ったときの Web 頼み。goo で検索すると http://hippo.ail-inc.co.jp/air/setup/netscape.html に日本語化のやり方が書いてあるのを発見。ja-netscape-3.04-1998-0115.tar.gz を/usr/ports/japanese/で展開、make すると良いらしい。早速 tar xzvf ja-netscape-3.04-1998-0115.tar.gz で展開して/usr/ports/japanese/netscape3.language/Makefile を読むと NLS.tar.gz,Netscape-JPfont,Netscape-v301-ja_euc.ad.gz が要ることが判明。http://www.bpel.tutics.tut.ac.jp/ take/Netscape/resources/ から拝借して、/usr/ports/distfiles/ に突っ込む。/usr/ports/japanese/netscape3.language/ に移って make。かりかりかり。あ、何かエラーメッセージ出しながら終った。

```
Warning: "/usr/ports/japanese/k12" non-existent
Warning: "/usr/ports/japanese/k8" non-existent
```

だそうだ。無いものは取ってくる。ftp://ftp.ee.uec.ac.jp/pub/FreeBSD/ports-2.2.5/japanese/ から持ってきて、もう一回 make。それそれ。今度は何も言われずに終了。安心して make install。これまた無事終了する。Command not found とか言われないように祈りながら netscape を立ち上げる。やや時間があってウィンドウが出てくる。どうやら成功したらしい。... あ、でもボタン群が変だ。普段なら File とか Options とか表示されるはずなのに文字化けしてる。リソースが変なのかな、と思って

/usr/local/lib/netscape/ja_JP.EUC/Netscape を眺めてみるが、何の解決にもならない。ここでも困ったときの Web 頼み。前出の http://www.bpel.tutics.tut.ac.jp/ take/Netscape/ をもう一回読んでいると、/usr/local/lib/netscape/nls/ja_JP.ujis を/usr/local/lib/netscape/nls/C にすれば良いらしいことを突き止める。
ということで、既存の/usr/local/lib/netscape/nls/C を改名・退避させて
cp /usr/local/lib/netscape/nls/ja_JP.euc /usr/local/lib/netscape/nls/C。さて、三度目の正直となるか、それとも二度あることは三度あるのか、netsacpe を立ち上げる。... 出てきたボタン群には日本語で「ファイル」とか「オプション」と書かれていた。何か見慣れないので気持ち悪いが、無事にインストールできたらしいことは確かである。がんがん Web 上を駆け回ってくれるし。良かったねぇ。

1998 年 3 月 9 日 日付と時間を直そう

久しぶりに部室に人が来た。そんな中、e さんに「カーネルの再構築をしてないの」と聞かれる。それまでカーネルには恐れ多くて触れられなかったので「まだです」と答える。再構築するとメモリの実使用量が増えたり、ブート時間が短縮したりと幸せなことが起こるらしい。とは言え、今からやってたら終るのが夜中になるのでカーネル再構築は後日になった。そんな折、e さんに日付と時間表示が腐ってることを訴える。e さん曰く、BIOS が変なのだそうだ。私は PC98 ユーザーなので、この辺の知識は全く無い。ともかく shutdown して BIOS 設定画面を呼び出す。で、e さんが日付やら時間やらを正常値に直していく。設定終了。日付がちゃんと Mar 21 になってる。今まで Jan 6 とかほざいてたとは思えないな。

1998 年 3 月 21 日 電通大、停電す

部室の扉の前まで行って気が付いた。今日は停電だ。しかも PC の電源は付けっぱなしだ。UNIX マシンは shutdown せずに電源を落すと不調をきたすことがあるのだ。こんなことなら前日に電源を落しとくんだった。今さら悔やんでも遅いけど。

1998 年 3 月 23 日 停電復活

今日、電通大は停電から回復した。部室に行き、kou の電源を入れる。home のマウント元の mola は t さんが立ち上げたらしい。あ、xdm が上がらない。仕方がないので X は使わずに login する。ふと、startx とかやったら X が立ち上がるんだろうかと思ってやってみたら、何とかというファイルを消したら上がるよ、と言われる。ものは試しである。言われたとおりにして startx。あら、上がった。この分なら xdm も上がるだろうと思ってリブートする。ほら、見慣れた xdm の login 画面が出てきた。それから思い出したように/sbin/fsck をかましてみる。ファイルシステムに大した問題は出なかったらしい。素の FreeBSD に限りなく近いからだろうか。

1998 年 3 月 27 日 百萬石に載せよう

さて、ここまで何だかんだと書き連ねてきたが、全ては MMA の会報「百萬石・'98 新歓版」に載せる話のネタのためである (と同時に自分が PC-UNIX をインストールするときに参考にするためだったりもする)。初心者のほほえましい (?) インストール作業なので間違いはかなりあるだろうが、その辺は笑って読み流して欲しい。さらっとね。ちなみに、やっておくと望ましいけどまだ手を付けてない作業・カーネルの再構築であるが、まぁいつになるかは分からないけど、そのうちにやってみるとしましょう。

gccの最適化のひみつ

清水丁

1998年4月2日

1 Intoroduction

gcc には最適化オプションがたくさんありますが、その効能一つ一つについてはあまり詳しく把握せずに -O2 や -O3 まかせになってしまう場合がよく見られます。

普通に使う分にはそれでも構わないのですが、コンパイル的な問題 (-O2 だと動くけど -O3 だと動かなくなった! 等) にぶつかった時は、最適化オプションによる効果と弊害について知ってると知っていないでは問題解決に大きな差がでるでしょう。

というわけで、今回は[1] gcc の最適化、最適化オプションについてつらつらと解説してみましょう。

2 基本的な最適化

1 定数乗算の展開

定数乗算の展開とは、古くから知られている最適化の方法です。乗算命令のある CPU でも、小さな数なら、乗算をシフトや足し算命令に展開するほうが一般的に速くなります。そのことを利用した最適化です。

具体的には、a*2 を計算するのに、multi 2,a などといった命令を使わずに add a,a などというような命令に展開することをします。

展開をするコンパイラと定数乗算を展開しないコンパイラの比較として、NEWS-OS4.2 附属の /usr/bin/cc と、gcc-2.7.2 の出力を比べてみることにします。

元のソース

```
test0.c:
        int test2(int a){       return 2*a; }
        int test3(int a){       return 3*a; }
        int test5(int a){       return 5*a; }
        int test10(int a){      return 10*a; }
        int test50(int a){      return 50*a; }
        int test100(int a){     return 100*a; }
```

gcc-2.7.2 でコンパイル。コンパイルオプション -O1 -S

```
        # 引数 a は $4 レジスタに格納されています。
        # 戻り値は $2 レジスタにて返ります

test2:
        j       $31             # いきなり return ... が、
                                # それといっしょに下の命令が
                                # 遅延スロットにて実行される
        sll     $2,$4,1         # 左に 1 ビットシフト (すなわち*2)

test3:
        sll     $2,$4,1         # $2 = $4*2
        j       $31
        addu    $2,$2,$4        # $2 = $4 + $2
```

[1] 時間的余裕があれば、各 CPU についての対比、gcc と egcs による出力コードの比較なんてことまでやりたかったのですが、迫るメ切、よどみなくやってくる仕事、上限なく増える TODO List の前にあえなく time up となりました。ネタ的にはもったいないので次回で補完したいです。

```
                                #(で、$2 は見事 a の 3 倍)

test5:
        sll     $2,$4,2         # $2 = $4*4
        j       $31
        addu    $2,$2,$4        # $2 = $4 + $2
                                #(で、$2 は見事 a の 5 倍)

test10:
        sll     $2,$4,2         # $2 = $4*4
        addu    $2,$2,$4        # $2 = $4+$2
                                #(で、$2 は a の 5 倍)
        j       $31
        sll     $2,$2,1         # さらに 2 倍して 10 倍

test50:
        sll     $2,$4,1         # $2 = $4*2
        addu    $2,$2,$4        # $2 = $2 + $4
                                #($2 は a の 3 倍)
        sll     $2,$2,3         # $2 = $2 * 8
                                #($2 は a の 24 倍)
        addu    $2,$2,$4        # $2 = $2 + $3
                                #($2 は a の 25 倍)
        j       $31
        sll     $2,$2,1         # $2 = $2*2
                                #(25 倍を 2 倍して 50 倍)

test100:
        sll     $2,$4,1         # $2 = $4*2
        addu    $2,$2,$4        # $2 = $2 + $4
                                #($2 は a の 3 倍)
        sll     $2,$2,3         # $2 = $2 * 8
                                #($2 は a の 24 倍)
        addu    $2,$2,$4        # $2 = $2 + $3
                                #($2 は a の 25 倍)
        j       $31
        sll     $2,$2,2         # $2 = $2*4
                                #(25 倍を 4 倍して 100 倍)
```

すべて綺麗にシフトと加算に展開されています。しかも gcc のほうは遅延スロットまで有効利用して無駄な時間を省いています。

NEWS-OS4.2 の cc でコンパイル。コンパイルオプション -O2 -S

```
        # 引数 a は $4 レジスタに格納されています。
        # 戻り値は $2 レジスタにて返ります

test2:
        mul     $2, $4, 2       # × 2
        j       $31
test3:
        mul     $2, $4, 3       # × 3
        j       $31
test5:
        mul     $2, $4, 5       # × 5
        j       $31
test10:
        mul     $2, $4, 10      # × 10
        j       $31
test50:
        mul     $2, $4, 50      # × 50
        j       $31
test100:
        mul     $2, $4, 100     # × 100
        j       $31
```

なんかもう見るに耐えません。ちなみに MIPS R[34]000 では、mul 命令は 10〜数十クロックかかります。

命令数で言うと、NEWS-OS の cc のほうが少なく見えますが、クロック数で言うと実は gcc のほうが速いコードを出しているのです。MIPS の mul 命令はちょっとクセモノで、乗算に時間がかかった場合、パイプラインの流れをストールすることによってパイプラインの同期を合わせてしまいます。mul 命令を使うより、シフト命令と加算命令に展開したほうがパイプラインのストールも無く、何倍も速いのは言うまでもありません。:)

... と、けなしまくった後、ほんとにここまで酷いかなぁ? とちょっと不安になって調べたら、どうやら NEWS-OS の CC は、最適化を一部 as に任せてしまっているようです。つまり、上のようなソースでも、定数のかけ算は as が shift や加算に展開してくれるのでした。

cc -c してできたオブジェクトを disassemble したもの。

```
main:
        jr      ra
        move    v0,zero
test2:
        jr      ra
        sll     v0,a0,1
test3:
        sll     v0,a0,2
        jr      ra
        bu      v0,v0,a0
test5:
        sll     v0,a0,2
        jr      ra
        addu    v0,v0,a0
test10:
        sll     v0,a0,2
        addu    v0,v0,a0
        jr      ra
        sll     v0,v0,1
test50:
        sll     v0,a0,2
        subu    v0,v0,a0
        sll     v0,v0,3
        addu    v0,v0,a0
        jr      ra
        sll     v0,v0,1
test100:
        sll     v0,a0,2
        subu    v0,v0,a0
        sll     v0,v0,3
        addu    v0,v0,a0
        jr      ra
        sll     v0,v0,2
        nop
        nop
```

ちゃんと展開してくれてました。ちょっと安心。そういえば AS が最適化してくれるのは RISC を売りだしたときのベンダの自慢 (?) でもありました。しかしもはや RISC のアセンブラを手で書かなければいけない状況というのはほとんど無い[2] と思いますが...。

2 定数演算の畳み込み

定数演算の畳み込みとは、いわゆる、3*a + 4*a などといった計算を 7*a と変換するものです。

[2] それでもまだ、手で書かないとやっていけない職場があるのは事実 (ゲーム開発とか)。でもそういう状況では AS の最適化に頼る程度のコード書きでは使いものにならないのも事実。

今ではコンパイラがこんな最適化は当然のごとくやってくれるので、見易さを優先するために 3*a + 4*a なんて書き方をしますが、コンパイラによっては書いた通りのコードを出力してしまうので注意が必要です。:-P

さすがに gcc は 3*a + 4*a なんて腐ったコードは、たとえ -O0 が指定されていても吐かないようで (さすがですな)、またもや比較対象がないので、ここでは、おバカなコンパイラとして (またもや) NEWS-OS4.2 附属の /usr/bin/cc を引合いに出してみます。

```
test.c:
        int test(a)
        int a;
        {
                return 3*a+4*a;
        }
```

NEWS-OS4.2 の cc でコンパイル。コンパイルオプション -O2 -S

```
test:
        mul     $14, $4, 3      # $14 = $4 * 3
        mul     $15, $4, 4      # $15 = $4 * 4
        addu    $2, $14, $15    # $2  = $14 + $15
        j       $31
```

腐ってますね。せめて mul $2, $4, 7 くらいのコードは出してくれてもバチはあたらないと思うんですが、やってくれません。

もしかしてまたアセンブラが最適化してくれるのか? と思って object まで吐かせてディスアセンブルして確認してみましたが、定数乗算をシフトと加減算に展開してくれるに留まり、さすが ×7 を計算するというアクロバットはやってくれませんでした。

```
test:
        sll     t6,a0,2         # n*4
        subu    t6,t6,a0        # n*4-n で n*3 を作る
        sll     t7,a0,2         # n*4 を作る
        jr      ra
        addu    v0,t6,t7        # n*3 + n*4
                                # ↑ as が遅延スロットを使ってくれる
```

一方 gcc はどんなコードを吐くかというと、

gcc-2.7.2 でコンパイル。コンパイルオプション -O2 -S

```
test:
        sll     $2,$4,1         # $2 = $4*2     ( $2 = 2*a )
        addu    $2,$2,$4        # $2 = $2 + $4 ( $2 = 3*a )
        sll     $4,$4,2         # $4 = $4 * 4  ( $4 = 6*a )
        j       $31             # return .
        addu    $2,$2,$4        # $2 = $4 + $2 ( $2 = 7*a ) 遅延スロット利用
```

ちゃんと畳み込みを行って直接 a を 7 倍してくれています。さらに、前節で説明したように、乗算命令を使わずに定数乗算を展開してくれています。もはや文句をつけるところはありません。

3 共通部分式の削除

共通部分式というのは、同じ計算式が何回も出現する際に、一回だけ計算しておき、その結果と使いまわす、というものです。

```
void test(int a,int b)
{
        func1(a*b);
        func2(a*b);
}
```

というような場合に、a*b の計算を一回だけ行い、その結果を func1() と func2() に渡すというものです。

gcc -O0 でコンパイルしたものと gcc -O1 でコンパイルしたものを以下に示します。

gcc -O0 でコンパイル

```
_test:
        movl 4(%esp),%eax       # eax = a
        imull 8(%esp),%eax      # eax = a*b
        pushl %eax
        call _func1             # func1(a*b)
        addl $4,%esp

        movl 4(%esp),%eax       # eax = a
        imull 8(%esp),%eax      # eax = a*b
        pushl %eax
        call _func2             # func2(a*b)
        addl $4,%esp
        ret
```

-O0 の最適化無しな場合だと、a*b という掛け算を二回も行うという無駄をしています。

gcc -O1 でコンパイル

```
_test:
        pushl %ebx              # ebx 退避

        movl 8(%esp),%ebx       # ebx = a
        imull 12(%esp),%ebx     # ebx = a*b

        pushl %ebx
        call _func1             # func1(ebx)
        pushl %ebx
        call _func2             # func2(ebx)
        addl $8,%esp

        popl %ebx               # ebx 復帰
        ret
```

a*b の結果を ebx レジスタに格納し、func1, func2 の引数に ebx を入れているのがわかります。

ただ、この共通部分式の削除という最適化は、その共通部分式部分が外部に影響を与えないというのが条件となります。つまり、

```
func1( pi(10) );
func2( pi(10) );
```

という場合は、たとえ pi(n) が 3.14 の N 倍を返す単純な関数であっても、pi() が外界に影響を及ぼさないということを gcc は知らないので、これは共通部分式として削除されません。

もっと極端な例で言うならば、

```
func1( bell(10) );
func2( bell(10) );
```

というような、10 秒間 beep 音を鳴らす関数を引数に持つ場合のことを考えてみればわかるでしょう。これが最適化されてしまうと、ベルが一度しかならなくなってしまい、意図してることと違うプログラムができあがってしまいます。

4 メモリ上の計算のレジスタ割り当て

いわゆる、gcc の -fforce-mem オプションで適用される最適化です。これにより、メモリ上の変数の計算を、なるべくレジスタ上でもって計算しようとします。

```
void test(x,y)
int *x,*y;
{
        *x = *y * (*x + *y) + *y + *x * *x + *x;
}
```

これを -fforce-mem したものと、-fno-force-mem したもので比べてみると

force-mem 最適化無し

```
_test:
        movl 4(%esp),%ecx       # ecx = x
        movl 8(%esp),%edx       # edx = y
        movl (%ecx),%eax        # eax = *x
        addl (%edx),%eax        # eax += *y     eax = X+Y
        imull (%edx),%eax       # eax *= *y     eax = Y(X+Y)
        addl (%edx),%eax        # eax += *y     eax = Y(X+Y) + Y
        movl (%ecx),%edx        # edx = *x      edx = X
        imull %edx,%edx         # edx *= edx    edx = X*X
        addl %edx,%eax          # eax += edx    eax = X*X + Y(X+Y) + Y
        addl %eax,(%ecx)        # *x += eax     X += X*X + Y(X+Y) + Y
        ret
```

*x や *y を参照するたびにレジスタ間接アドレッシングを行っていることがわかります。すなわち、変数一つ見るたびにメモリにアクセスしに行ってしまう!!

force-mem 最適化あり

```
_test:
        pushl %ebx
        movl 8(%esp),%ebx       # ebx = x
        movl 12(%esp),%eax      # eax = y
        movl (%ebx),%ecx        # ecx = *x
        movl (%eax),%edx        # edx = *y

        leal (%edx,%ecx),%eax   # eax = x+y     eax = X+Y
        imull %edx,%eax         # eax *= edx    eax = Y(X+Y)
        addl %edx,%eax          # eax += edx    eax = Y(X+Y) + Y
        movl %ecx,%edx          # edx = ecx     edx = X
        imull %ecx,%edx         # edx *= ecx    edx = X*X
        addl %edx,%eax          # eax += edx    eax = X*X + Y(X+Y) + Y
        addl %ecx,%eax          # eax += ecx    X += X*X + Y(X+Y) + Y
        movl %eax,(%ebx)        # *x = eax      *x = X;
        popl %ebx
        ret
```

こちらは、*x や *y をあらかじめレジスタに格納し、

```
X = *x;
Y = *y;

Y*(X+Y) + X*X + Y + X
```

を計算しています。その間すべてレジスタ演算だけで、メモリアクセスは皆無。最近ではいくら 2 次キャッシュまで付いているとは言え、メモリとレジスタじゃぁアクセスの速さには雲泥の差があります。force-mem したほうが実行速度が速いのは言うまでもないでしょう。

5 ループ展開

インストラクションキャッシュのサイズによっては、ループを展開すると逆に遅くなるなんてこともありますが、最近はキャッシュ容量の肥大化にともなって、ちょっとしたループはやっぱり展開したほうが速いです

私が昔 X68030 でアセンブラを書いてた頃は、ループ展開後、そのコードがキャッシュの同一タグ内に収まるようにアライメントを調節するという細くて泥沼的な配慮をしながらやっていましたが、昨今のマシンではどうなんでしょう? 何も考えずがんがん展開しても、数百 kb オーダーのキャッシュのおかげでぜんぶヒットしてしまいそうな気もします。

ちなみにこのオプションは -O3 でも有効にならないようです。gcc の manpage には -O2 で有効になると書いてありますが、-O2 でも -O3 でも有効になりませんでした。

```
void test()
{
        int i;
        for(i=0;i<10;i++){
                func();
        }
}
```

gcc -O2 でコンパイル

```
_test:
        pushl %ebp
        movl %esp,%ebp
        pushl %ebx
        xorl %ebx,%ebx
        .align 2,0x90           # アライメント合わせ!!?
L5:
        call _func
        incl %ebx               # i++
        cmpl $9,%ebx            # i<=9
        jle L5

        movl -4(%ebp),%ebx
        leave
        ret
```

loop がキャッシュに有効に載るようにいちおうアライメントを合わせています (たぶん)。ここらへんはさすが gcc といった感があります。

gcc -O2 -funroll-loops でコンパイル

```
_test:
        pushl %ebp
        movl %esp,%ebp
        call _func
        call _func
        call _func
        call _func
        call _func
        call _func
        call _func
        call _func
        call _func
        call _func
        leave
        ret
```

綺麗に展開されています。さて、Pentium ではどちらが速いでしょう? (func() の内容にもよりますが)

6 関数のインライン展開

いわゆる、小さな関数をマクロのように扱ってしまうものです。単純に展開するわけではなく、ちゃんとレジスタの割り振りまで考えて展開してくれるので、ちゃんと使いこなせば実はマクロ以上に便利です。

このオプションは gcc -O3 以上でないと有効になりません。

```
static int func(int a)
{
        int r;
        r = func1(a);
        return func1(r);
}

int test1(int x)
{
        return func(x*2);
}
```

gcc -O2 でコンパイル。

```
_func:
        pushl %ebp
        movl %esp,%ebp
        pushl 8(%ebp)           # func() の引数を積んで、
        call _func1             # func1() を呼んで..
        pushl %eax              # その返り値でもういちど、
        call _func1             # func1() を呼ぶ
        leave
        ret

_test1:
        pushl %ebp
        movl %esp,%ebp
        movl 8(%ebp),%edx
        leal 0(,%edx,2),%eax
        pushl %eax
        call _func
        leave
        ret
```

いたって普通のコーディングです。一方 inline-functions だと...

```
_test1:
        pushl %ebp
        movl %esp,%ebp
        movl 8(%ebp),%edx
        leal 0(,%edx,2),%eax
        pushl %eax
        call _func1
        pushl %eax
        call _func1
        leave
        ret
```

見事に func() が展開されています。さらに、static と宣言されている func() 自体も無くなってしまいました。static と宣言されていても、inline 展開された関数が無くなってしまうのは具合が悪い場合があるので、そんな時は -fkeep-inline-functions を指定すれば、関数本体もそのままコードとして残されます。

7 スタックフレーム作成についての最適化

スタックフレームとは、関数内でローカル変数を使う際に使用されるスタック上のワークのことです。そしてそのスタックフレームの最後を指すポインタがフレームポインタです。

スタックが下の位置にある時、

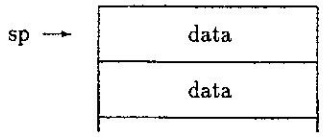


図 1. stack になんらかのデータが積まれている

この状態で関数コールを行うと

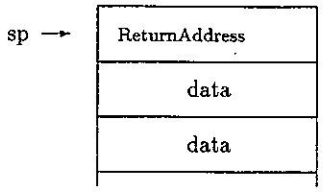


図 2. return address が積まれて関数の場所へ jump!

で、関数内でローカル変数を使う場合、

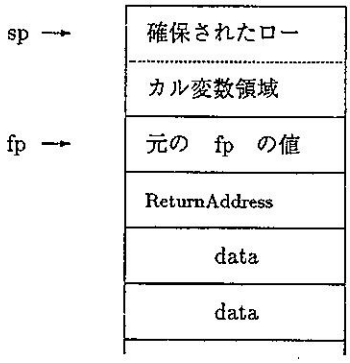


図 3. スタックフレームを確保したところ

このローカル変数領域がスタックフレーム、そして元の sp をセーブしたものがフレームポインタとなります。

ローカル変数がある場合はもちろんこうするしか無いのですが、gcc ではローカル変数を確保する必要のない場合も、サイズ 0 のローカル変数領域を作成し、スタックフレームを作ります。なぜそのような無駄なことをするかというと、その理由は デバッグのためにあります。

gdb の breakpoint 機能などを使って処理を止めた後、where などのコマンドを使うとどの関数から呼ばれたかがわかりますが、それはフレームポインタ+4 の位置にある、自分を呼んだ関数へのリターンアドレスを参照してその機能を実現しています。逆に言うと、フレームポインタを使っていなければ、スタック上のリターンアドレスが格納されている位置がわからないため、where 機能は使えません。

つまり、関数内の任意の位置で、自分を呼んだ関数 (内を示すリターンアドレス) を知るために、フレームポインタを使っているのです。

これを抑制するオプションが、omit-frame-pointer です。必要なローカル変数領域のサイズが 0 な場合はフレームポインタを使用せず、スタックポインタだけを使うようになります。ただ、このオプションは上で説明した通り、デバッグを大変困難にするものなので、gcc では -O3 (以上) を指定しないとこのオプションは有効になりません。

以下に -fomit-frame-pointer を指定した場合と指定しなかった場合のソースを示します。

C のソース

```
void test(int a)
{
        test2(a);
        test2(a);
}
```

-fomit-frame-pointer を指定しなかった場合。

```
_test:
        pushl %ebp              # frame pointer 退避
        movl %esp,%ebp          # frame pointer を作る
        pushl %ebx              # ebx 退避

        movl 8(%ebp),%ebx       # ebx = test() の第一引数

        pushl %ebx              # スタックに積んで
        call _test2             # call
        pushl %ebx              # スタックに積んで
        call _test2             # call

        movl -4(%ebp),%ebx      # ebx 復帰
        leave                   # frame pointer 復帰
        ret                     # return
```

この場合、frame pointer を作る動作は実はまったくの無駄です。

次に gcc -fomit-frame-pointer でコンパイルした場合です。

```
_test:
        pushl %ebx              # ebx 退避

        movl 8(%esp),%ebx       # ebx = test() の第一引数

        pushl %ebx              # スタックに積んで
        call _test2             # call
        pushl %ebx              # スタックに積んで
        call _test2             # call

        addl $8,%esp            # スタック補正

        popl %ebx               # ebx 復帰
        ret                     # return
```

フレームポインタを使うことをやめたかわりにスタック補正が必要となり一命令多くなりますが、それでもフレームポインタを作るためのコードが減るので、全体的にはこちらのほうが速くなっています。

A 参考文献

- 大昔の Oh!X の gcc の記事 (記憶を頼りに)
- gcc version 2.7.2.2+myc1
- NEWS-OS 4.2R の /usr/bin/cc

B 次回予告?

- その他の option について
- マシン固有な最適化
- egcs による拡張

MMA UNIT TIPS

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

1 まえふり

とりあえず、今回もお茶を濁す記事です。
赤入れしてもらう時間が無かったので、改定や増補は HTML 版でやろうかと思います。間違いや追加してほしいものがあればメールください。

2 commands

1 URL を指定してファイルを取得する

ftp スキームのみならば ncftp 、http スキームも含む場合は wget, lwp-download(libwww-perl), httpdown などのコマンドで可能です。
基本的にコマンドラインで使用する場合、

```
%%wget http://delegate.uec.ac.jp:8081/club/mma/index.html
```

とういうような形で使用します。
それぞれ proxy に対応しているんで、適切に設定してやればプライベートネットワーク内部からもちゃんと使えます。再帰的に持ってこれたり、パイプに喰わしたりもできます。詳しくはは man 読んでください。
MMA の環境（正確には mola ）では wget がプロキシを参照するように設定されているので、wget を使えば良いでしょう。ただし非インタラクティブな動作しかできないので、インタラクティブに ftp したいなら、ncftp が delegate 経由でのアクセスに対応しているらしいので頑張って設定してください。まぁ、netscape や lynx でもよいでしょう。
また、もう少し凝った動作をさせたいなら perl と libwww-perl モジュールを使えばいろいろできます。
詳しくは lwp-download のソースや/usr/local/lib/perl5/man/man3/LWP.3 を参考にしてください。

2 ftp のおまじない

tips というほどではないですけど、classical ftp の基本のおまじないです。

```
ftp> bin #バイナリモード
200 Type set to I.
ftp> prom # mget コマンドでいちいち聞いてこなくなる
Interactive mode off.
ftp> hash #転送の状態を表示
Hash mark printing on (1024 bytes/hash mark).
```

3 ftp で再帰的にファイルを取得

これも tips というほどではないですけど、ncftp2 のコマンドラインでもインタラクティブモードでも再帰的にファイルをもってこれます。get に -R オプションをつけてみましょう。
classical ftp でもローカルにリモートと同じ構成でサブディレクトリが掘ってあれば、mget でも再帰的に持って来れます。
あと、ftp server が wu-ftpd などで、tar や compress に対応していれば、ファイルやディレクトリごと持って来れる事があります。login 時のメッセージを確認して見ましょう。たとえば、freebsd の mirror site だと usr.bin/lex に対して

```
ftp> cd usr.bin
ftp> get lex.tar.Z
```

できる、というような例が FreeBSD ハンドブックにあげられています。

4 ファイルのコピー

ディスクを増設してパーティションの内容を移動させたいときなど、元の状態を保持したままコピーしたいという場合のはなしです。

BSD cp (4.3BSD 以降?)

```
cd fromdir; cp -pRP . todir
```

ハードリンクが駄目、それぞれ別ファイルに分裂

GNU cp

```
cd fromdir; cp -a . todir
```

デバイスファイルが駄目

tar

```
cd fromdir; tar cf - . | tar CxfBp todir -
```

で、シンボリックリンクやパーミッションをそのままでコピーできます。まぁ、デバイスファイルまでコピーすることはあんまり無いので、GNU cp が一番お手軽で良いのではないでしょうか。

5 ベタテキストを印刷する時は a2ps

ベタテキストを印刷する時は a2ps を使いましょう。可読性も上がるし、紙の節約にもなります。たぶん

6 メールにフィルターをかます場合の注意

フィルターが暴走した場合に mail explosion を防ぐために、

```
"/usr/home/hoge/script/filter || exit 75 #YOUR_LOGIN_NAME"
```

なんてやってあげると、エラーを起こした場合に sendmail がすぐにエラーメールとして処理せずに一時的にキューに貯めてくれるのでよいです。

もう少し凝ると、procmail の sample のように、

```
"|IFS=' ' && p=/usr/local/bin/procmail \
                && test -f $p && exec $p -Yf- || exit 75 #YOUR_LOGIN_NAME"
```

という風にもできます。

3 shell

1 リモートログインのサスペンド

rsh,ssh などのリモートログインアプリも ~^Z でサスペンドできます。リモートで作業しているときに、ちょっとローカルでなにかしたいなぁ、なんて時に便利です。

もう少し詳しく書くと、たとえば、ホスト A からホスト B に、ホスト B からホスト C にリモートログインして作業している場合を考えてみましょう。

~ (チルダ) を一回押して ^Z (Ctrl+Z) でサスペンドすると、ホスト A の shell に戻ります。そこで fg すると、また ホスト C の shell に戻ります。

また、~ (チルダ) を二回押して ^Z (Ctrl+Z) でサスペンドしてやると、ホスト B のシェルに戻れます。

あと、ときどき ~ の入力をリモートログインアプリが取りこぼすことがあるので、そのときは一度改行してもう一度 ~ を入力すればよいでしょう。

それから、サスペンドしている間は pty (疑似端末) の制御も切れるためか[1] サスペンドしている先で talk とかかかってきててもわからないので注意が必要です。

[1] ちゃんとした理由については「UNIX ネットワークプログラミング」(トッパン) の "リモートログイン" の章あたりを参照のこと

4 X

1 キーボードによるペースト

kterm や xterm のリソースを設定してやることによって、特定のキーに X のペーストの機能[2] を割り付けることができます。

適当な設定ファイル[3] で以下のようなリソースを設定してやると Function 12 キーでバッファからペーストされます。

```
KTerm*VT100*translations:       #override \
<Key>F12: insert-selection(PRIMARY, CUT_BUFFER0)
```

これで Elbereth [4] とダンジョンの床に刻むのも楽になります。Wizard 派の人には涙ものです (^^; 三年前に知ってれば...

2 フォント

X に付属する日本語フォントは少なく[5] 、環境や宗派[6] によってはつらいものがあります。とりあえず、フォントを追加する方法はいくつかあります。

- ベクトルフォントに対応した X Server を利用する
  - X-VFlib、X-TrueType を使用
  - winodws 上で ASTEX-X の built-in-ttf を使用
- 各種フォントをコンバートして好みのビットマップフォントを用意する
  - windows 上で msf2bdf を使って TrueType font から直接 bdf を生成
  - fontx2 フォントを WFontX 等で生成するなどして用意し、それから fontx2bdf で bdf を生成

といったところでしょうか、ただし注意点としては、いくらベクトルフォントを使っていても、12 dot 以下のサイズのフォントを生成する場合、ベクトルフォントによって視認性がかなりかわります。いろんなフォントを試して見て下さい。あと、フォントコンバータの類は NIFTY にしか置いてないみたいです。

5 最後にひととして

- 真実は最大の武器、正直は最良の戦術
- MMA において "誰か" とは somebody ではなく nobody である
- 有言実行 >> 有言不実行 >> 無言実行 >> 無言不実行
- 和して同じず

---

[2] X のコピー&ペーストの機能に関しては「入門 X Window」(ASCII) に詳しい、詳細はそちらを参照のこと

[3] 一般には .Xdefaults とか .Xresources ですが、そこの環境に依存するので周りの人に聞いて見ましょう

[4] トールキンの指輪物語の神様、NetHack でこの神の御名を刻むと、魔を退けることができる。スポイラーの訳注によると "エルベレス (Elbereth) とは指輪物語の舞台である中つ国の神々であるヴァラの一人、ヴァラダの別名で「星々の女王」という意味である。彼女の名は聖なる力を帯びており、指輪の仲間であるサムワイズは彼女の名前を唱え、キリス・ウンゴルに住む大蜘蛛シェロブをしりぞけた。" だそうである。

[5] 14,16,24 dot のみ

[6] VGA のノートとの人とか、丸文字以外のフォントを取りこぼしてしまう人とか

CCさくらのぱろでぃ
もしくは
だめのありよう

ottan

1　きゃすと

さくら	uirou
藤隆さん	tree
桃矢	mama
雪兎	oolong
知世	sanjigen
山崎君	fujita
ケロ	ottan
目つきの悪い兎 (クロウカード JUMP)	imai
クロウカード 破壊	shimizu
クロウカード 麦酒	rikachan

2　たたかい

目つきの悪い兎と対峙するお約束の幼児向けブランドものを着込んだ さくら
(”さくら萌え〜”というだめにんげん以外には聞こえない波動で麦酒のカードのたましいの叫びが夜のしじまにに響き渡っていきます)

さくら　『やっと 追い付いた... わ』(恥ずかしげに、こぶしを効かせて”ぐっ”)
『いま~H~H... あなたのセクトはもう消滅しちゃったし、おとなしくカードに戻ってくれたら嬉しいなぁ』
(目つきの悪い兎 椅子を近くにあった蹴飛ばします、”がすっ~2”)

ケロ　『ダメじゃないですかそんなんじゃ、さくら、これで行きましょう』
(ケロちゃん書きかけの修論をとりだします)

さくら　『はぁ、そうですね』
(さくらプリントアウトされた修論をかまえます)
(呪文詠唱省略)
『汝のあるべき姿に戻れ！、「だめにんげん」!! 』
(修論に打ちのめされて、だめぶらっくに堕落した目つきの悪い兎は職場に向けて退場されます)

知世　『ダメダメでしたよ、さくらちゃん、QV-10 に勇姿はおさめたんで、web にのっけてあげますよ』

さくら　『恥ずかしいからやめてって言っても、だめなんだよねぇ？』

ケロ　『きまってるじゃないですか』

さくら　『はぁ〜.....』

ケロ　『もう１枚のこってるはずだから、ちゅういしてくださいね』
(そこに、響き渡る大量の CD が砕け散る音、破壊のクロウカード登場)

知世　『こんどはかっこ良く決めてくださいね、さくらちゃん』

さくら　『はぁ、がんばってみます』
(とりあえず、呪文を唱えてみる さくら)

さくら　『ざーらーどぐろーる、えーへーどえい、まるとぅーばす、こーむど!!』
(同じクールではありますが、ちょっと効いて無いようです)

さくら　『いあいあいあ、いあ、よぐそとーす....』
(なんだかあたりに狂気が漂って来ました、これ以上唱えると危なそうです)

さくら　『…ｘａ…ｉｋｌｏ…ｓｅ…ａｕｓ…ｖｉ…ｌａｋｓ……ｐｏｅｔｌｏ…ｉｍｋ…ｑｏ…ｖａｌｄｕ……』
(破壊のカードがダメージをうけたようです、「ふるふるぅ～」「こくこくぅ～」なんてたましいの波動が聞こえてきます。いまのうちに封印してしまいましょう、えいっ!)

さくら　『汝のあるべき姿に戻れ!、「だめにんげん」!!』
(なんかのカードのちからで出現した大量の超自然の小動物たちといっしょにだめれっど退場です)

さくら　『はぁ、つかれた.....』

3　あさの風景

(さくらさん――藤隆さんはキャラクターものの抱き枕を抱えてまどろんでいます)

さくら || 藤隆さん　『はにゃ～んはふ～しあわせで死にそう～』

4　続あさの風景

さくら　『きゃーーー!遅刻しちゃうーーー!』
ぱたぱたぱた

桃矢　『なに ぱたぱた暴れてんだ』
(お兄さんはお気に入りの太田貴子が声優[1] をやってたキャラクターのキャラクターもののマグカップでコーヒーを飲んでいます)

藤隆さん　『おはよう、さくら』
(メイドロボットのキャラクター物のエプロン姿のお父さんが、クマのキャラクターものの茶碗とお椀にさくらさんの朝ご飯を用意してだしてくれます)

お話の途中ですが、血糖値が閾値をきってしまいました。おいしいチョコレート[2] が入手できなかったのでもう精神が持ちません。
借り物のマンガを叩きつけるわけにもいきませんし、もう一文節たりとも書きたくありません。

では、みなさん、さようなら。

りゃく、略、大幅に省略!!

5　えぴろーぐ

こうしてだめにんげんは駆逐され、
廃人戦隊ダメレンジャーは滅んだ。
ありがとう さくら!
きみの尊い犠牲をわたしたちは永遠に忘れない!!

THE END

[1] 今度 LD 出るみたいですね
[2] "メルティーキッス"は冬季限定発売です、北海道のおみやげも、スイスのおみやげもありません、ついでに言うとリンツは嫌いです

三次元画像処理ライブラリ製作

ikidou@mma.club.uec.ac.jp

1998/04/02

概 要

一年生終りのころ製作を思い立った三次元画像処理ライブラリが、やっとこさ動く段階になりました。良かった良かったうれしいなっ。で、どんなことができるのか（できないのか）を書いていきます。

1 三次元画像処理には何が必要か

おおまかに二つにわかれているとの考えで作成してきました。まず、生成、変形、加工といった、物体を操作する部分が一つ。もう一つが物体のデータを現実と矛盾なく画面に表示させる部分です。

1 物体の操作

基本的に以下のようなことができるようにしました。

- 物体の生成
- 物体に属性を設定できる (色や光の反射率等)
- 行列による変形
- 光をあてることにより現実感がだせる
- 物体のネスティング

最後の"物体のネスティング"は個人的に好きなところなので少し語らせてもらいます。格闘ゲーム[1] を考えてみましょう。キャラクタのデータとして、頭、体、腕、足等のパーツにわけて定義されています。これらのパーツそれぞれに変換行列を作用させることにより、まるで生きているかのごとく動かしているのです (おそらく、きっと、多分...)。

キャラクタが前進するためには、各パーツに前進の要素が含まれる変換行列があたえられます。しかし、突きの要素を含んだ変換行列は腕の部分にのみ[2] 作用します。もしこの変換行列が頭部に与えられたらどうなるでしょう? きっと突きのたびに、頭部がはずれて宙を舞うとってもシュールなゲームになってしまいます。このようなことをさけるため"物体のネスティング"[3] があるのです。

ちょうど人間の関節のように、頭→肩→上腕→前腕→手のようにパーツをリンクさせます。手を動かすときは前腕に対してどのように動くかを指定し、同じように前腕は上腕に、上腕は肩に、肩は体に対して動かします。そして、実際には体、上腕、前腕、手の動きを合成して、手の変換行列を作成するのです。

この考えを利用することにより、惑星運動、蛸の足といったものをシミュレートすることができます。

2 物体の表示

物体の操作にでは、全て三次元のデータとして計算をおこなってきました。しかし表示する画面は二次元であるので、いかにしてデータに矛盾することのない表示をするかがポイントです。ここでおこなったことは、

- 光による表示色の変化
- 遠近方を使用
- 面単位の Z ソートで前後関係をつくる
- 塗りつぶし法による隠面処理

塗りつぶし法とは、画面奥に存在している (ように見える) ポリゴンから描いていくことで、前面のポリゴンの方が優先的に表示されるようにするものです。この方法では表示される必要のないポリゴンを多量に描いてしまう欠点があるので、裏を向いているポリゴンは表示をしない[4] ようにしました。隠面処理には他にも Z バッファ法、スキャンライン法、レイトレーシング等がありますが、塗りつぶし法が自分にとって一番実現しやすかったので採用しました。

---

[1] もちろん 3D のやつッス。

[2] 下半身が死んでちゃ生きた突きは打てんとか、武術的ツッコミはやめてね。

[3] 正確に表現すれば、"座標系のネスティング"です。

[4] uirou さんのアドバイスによるものです。多謝。

2 自作自演のFAQ

Q1: 3DCG に興味をもったのはいつですか?

A1: びかびかの一年生のとき部室で uirou さんの作った[5] くるくるまわる 3D MMA logo を見たとき、個人の力でここまでできるようになるんだ、と素直に尊敬して[6] しまいました。いつかこんなプログラムが組めたらなあと考えたのを覚えています。一年の終りごろ、3D 技術を用いたゲームを作りたくなったのがきっかけで始めました。

Q2: 他に便利な 3D ライブラリがあるのになぜわざわざ作ったのですか?

A2: 単純にいってしまえば知らなかったの一言です。で、それらの存在をしったときには 3DCG や C 言語の勉強を始めてしまっていたので、半分意地、半分習作と割り切ってしまいました。[7]

Q3: 何か得るものはあったのですか?

A3: 非常に難しい質問です。得たものは

- プログラムをする喜びを知った
- 三次元画像処理の基礎知識を得た
- C 言語がしゃべれるようになってきたでちゅー
- 理論とそれを実現しようとしたときのギャップを知った

とあげることはできるのですが、失ったもの[8] も大きかったような気が...

Q4: 突然技術的な話ですが、物体に色をつけたり、光の色を変えたりしても全部灰色で表示されるのですが。

A4: あうっ、実はカラー表示にするとあまりにも多量の色を消費するため、表示段階で、グレースケールに変換しているのです。もしカラー化するならディザリングを使用しなければならないため、現在お勉強中でふ。

Q:5 物体の動画を表示させたらとっても重いです。何とかなりませんか?

A:5 行列演算に色計算、物体の変換、ポリゴン描写とばりばり CPU パワーを喰ってしまう処理が多く仕方無い部分なのですが、もう少しアルゴリズムをねって高速化してみます。

Q6: これからどんなことがしたいですか?

A6: まずはサクラ 2[9] を... もとい、カラー化とアニメーションを楽に作成できる工夫の追加、高速化、モデリングの工夫等々いろいろあります。3DCG と関係ないところではシステム管理を全く知らないので、そのあたりの勉強をしてみたいところです。でも今は

「走れー光速のーていこーく華激団ー」...

---

[5] 最初は sanjigen 先輩が作ったとばかり思っていました、alias からなんとなく...
[6] このときは sanjigen 先輩を
[7] 素直に使っていた方が幸せだったかも
[8] 時間とか、健康とか、視力とか、他にもうまくプログラムが組めなくて逃避したり、鬱っぽくなったり等々
[9] 現在 (これを書いているとき) が 1998/04/02 02:20 で、サクラ対戦 2 の発売 (1998/04/03) までもうすぐもうすぐ