MMA

Microcomputer Making Association

1995年 調布祭 百万石

目次

グループワーク支援システム ARGO	楯岡 孝道	1
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	楯岡 孝道	5
ハードボイルドとコンピュータ	4C boss	11
へなちょこ三次元ポリゴンプログラム	uirou	13
perl なスカラデータのひみつ	しみずりょう (仮名)	17
百万石スタイルファイルの作成	君島 秀征	23
'login' までの道	uncover	25

電気通信大学MMA

グループワーク支援システム ARGO

楯岡 孝道 (tree@mma.club.uec.ac.jp)

Abstract

ARGO は電子メールを用いたグループワークを支援する機構である。ARGO では議事録及び共同作成文書の保存/検索/取得、作業者名簿の更新等を全て電子メールを用いて行う事ができる。本文章では、ARGO の概要を述べる。

1 はじめに

数年前までごく限られた人々にしか利用されていなかったインターネットは、近年急激に普及し、特に広義のインターネット接続である電子メールは、パソコン通信などとの接続も考えれば、希望すれば誰でも利用できる状況になった。

MMA においても 1995 年 4 月 11 日の学内ネットワークへの IP 接続を行い、同 22 日には電子メールの運用が始まっている。

電子メールを利用することにより、時間と場所に依存せずに文章作成などの共同作業を行う事ができる。特に、議論を行う事のできる ML (Mailing List) の機構は広く普及しており、さまざまな実装が存在する。同様に、メールによるファイル管理を行うことのできる実装も存在するが、不必要な機能が多く、巨大なシステムになりがちである。

そこで、シンプルな実装として ARGO (Automatic Reliable Group-work Operator) システムを実装した。

2 システム概要

ARGO システムでは以下の 3 つの異なるメールアドレスを利用する。

- **ML アドレス**
  議論を行う ML のアドレス。このアドレスへ配送されたメールは、メンバー全員に再配送されると同時に、後の参照に備え記録される。
- **コマンドアドレス**
  ARGO システムに対するコマンドを与えるアドレス。このアドレスに配送されたメールは、ARGO システムに対するコマンドとして解釈、実行される。
- **管理アドレス**
  システムの管理者のアドレス。システムは異常を検知すると、このアドレスに通知を行う。

メンバー間の議論は、ML アドレスを用いて行う。議論のログはログ記録領域に通し番号をファイル名として保存される。この際、後の参照のために、通し番号とメールのサブジェクトを記録したインデックスファイルを作成する。

議論の結果作成されたファイルは、別途コマンドアドレスに登録コマンドを付加して送信する事により、ライブラリ記録領域に保存される。この際、ファイル番号とタイトルを記録したインデックスファイルを作成する。ファイルの内容はメールで扱えるテキストであればなんでも良い。

ARGO システムの操作は全てコマンドアドレスへのメール (以後、コマンドメール) によって行う。コマンドメールによって、ログ領域に対する参照、ライブラリ記録領域のファイルの取得/新規登録/更新/削除、ML のメンバーの登録/削除等が可能となっている。コマンドの実行結果はメールによって返信される。

コマンドメールでは、メールのサブジェクトでコマンド及び引数を指示する。メールの本文は全てコマンドの入力として用いられるが、コマンド以外の入力を必要としないコマンドに関しては無視される。

2.1 ログ参照

コマンドメールによるログ参照では、以下の操作が可能である。

- **LIST**
  メールの通し番号とサブジェクトを記録した

インデックスファイルを取得する。

- **GET**
  メールの通し番号を指定して、メール本文を取得する。

ログ領域の通し番号は、ML で配送されるメールに付加される通し番号と一致している。そのため、議論等においてこの番号を用いて一意に過去の議論のメールを参照する事が容易になっている。

2.2 ライブラリ操作

コマンドメールによるライブラリ参照では、以下の操作が可能である。

- **LIST**
  ファイル番号、マジックナンバー (後述)、タイトルを記録したインデックスファイルを取得する。

- **GET**
  指定したファイル番号のファイルの本体を取得する。

- **PUT**
  メールの本文を指定したファイル番号とタイトルで記録する。ファイル番号として “*” を指定する事により、未使用の番号を自動的に割り当てる。成功すれば、ファイル番号とファイルのマジックナンバーを返す。

- **UPDATE**
  指定したファイル番号のファイルを、メールの本文の内容に更新する。成功すれば、新たなマジックナンバーを返す。

- **REMOVE**
  指定したファイル番号のファイルを、ライブラリから削除する。

2.2.1 マジックナンバー

共同作業においては同一のファイルに対し、複数のユーザが更新を行うことがある。通常の共同作業環境ではこのような場合、ユーザは更新を行う前にファイルを排他的にロックし、ロックしたファイルの内容に対して更新を行った後に、ファイルのロックを解除する。ロック中のファイルには他のユーザは更新を行えないため、このようにすることにより、ファイル操作中に他のユーザの変更が加わる恐れなしに、更新を行う事ができる。

メールによる共同作業においても同様の機構を用意する事は可能であるが、メールの、到達性が常に保証されているわけではないという、特性のため、問題が生じる可能性がある。具体的には、ファイルのロックを行ったユーザが長期間メールの到達性を失った場合、他のユーザはそのファイルの更新を行えない状況に陥る。

RCS 等の一般の共同作業システムでは、この問題に対して、ファイルのロックを強制的に解除する機構を用意する事により対処している。しかし、メールではユーザによって遅延差が大きいため、どの程度の期間ロックされていれば異常であるという判断が困難である。

そこで、ARGO ではマジックナンバーを用いて緩い排他制御を行っている。

マジックナンバーは、ファイルの内容から算出された 6 桁の数字である。この番号は、ファイルの一意性を保証するために利用される。

ライブラリに対する操作のうち、ファイルの内容を書き換える UPDATE と REMOVE では、このマジックナンバーを指定する、コマンド実行の際には、実際のファイルから算出されたマジックナンバーと比較され、異なる場合には操作が拒否される。

マジックナンバーはファイルの内容毎に異なるので、これを指定する事により、ユーザが意図しないファイルに対して破壊的操作を行う事を防ぐ事が可能になる。同一のファイルに対して、複数の更新要求が行われた場合、最初の要求のみが実行される。最初の要求によりファイルの内容が更新されると、マジックナンバーが変化するため、他の要求は拒否される。これにより、他のユーザの更新内容を上書きしてしまう危険性を回避している。

2.3 ML メンバー管理

コマンドメールによるメンバー管理では、以下の操作が可能である。

- **SUBSCRIBE**
  指定したメールアドレスをメンバーリストに付加する。

- **UNSUBSCRIBE**
  指定したメールアドレスをメンバーリストから削除する。

- **LIST**
  メンバーリストを取得する。

メンバーリストには、配送形態、メールアドレスの他、本名等のコメントが登録される。配送形態には、通常の配送 (direct 配送) の他、定期的にまとめて配送する一括配送 (batch 配送) があり、自由に選択できる。両方の形態による配送を受ける事も可能である。

3 実装

ARGO システムは、次の 4 つの perl スクリプトから構成される。

- **argo.pl**
  コマンドアドレスに配送されたメールを処理する。

- **dist.pl**
  ML アドレスに配送されたメールを処理する。

- **pushmail.pl**
  ログ記録領域への保存及び、batch 配送用ファイルの作成を行う。

- **batchmail.pl**
  batch 配送を行う。

各スクリプト間のデータの流れは図 1 のようになる。

ML アドレスに配送されたメールは、dist.pl で通し番号等のいくつかのヘッダを付加され、direct 配送の宛先に再配送を行う。同時にメールの内容を pushmail.pl に伝達する。pushmail.pl では不必要なヘッダの削除、日本語の MIME ヘッダ [2] のデコード等を行った後、ログ記録記録領域への記録と、batch 配送用のファイルへの追加を行う。

batchmail.pl は cron により定期的に起動され、pushmail.pl によって作成された batch 転送用ファイルを batch 配送の宛先に配送する。

コマンドアドレスに配送されたメールは、argo.pl でコマンドとして処理され、結果がメールで返される。

各スクリプト間のデータの流れのうち、dist.pl から pushmail.pl に送られる情報以外は全てファイルを介して伝達される。このように、各部分が素な接続を行っているため、システムの一部分の改良や置き換えが容易になっている。

4 認証

本システムには認証機構が一切用意されていない、つまりメールアドレスさえ知る事ができれば誰でも全ての操作が可能になっている。これは特に参加者に制限を設けない共同作業を想定して設計されているためである。

悪意ある者により破壊活動を行う事は可能であるが、たとえ一般に行われている送信者アドレスによる認証を行ったとしても、現在のメール配送機構では悪意あるものの偽造を防ぐ事はできないので、あえて対処していない。

PGP [3] 等の電子署名、暗号化の技術を併用する事により、安全な認証を行う事も考えられるが、参加条件に制限を加えない以上は殆んど意味が無い。メールの署名をチェックし、通信内容を全て暗号化する安全な ML も研究課題としては面白いが、それは別の問題である。

5 現状と今後の予定

現在、実装を終了し、MMA 内 ML 実験マシンにおいて数人規模の試験運用を行っている。試験運用で細かい仕様を詰め、大きな問題が無ければ、実際に広域共同作業実験を行う予定である。

参考文献

[1] David H. Crocker, “STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES,” RFC822, 1982.

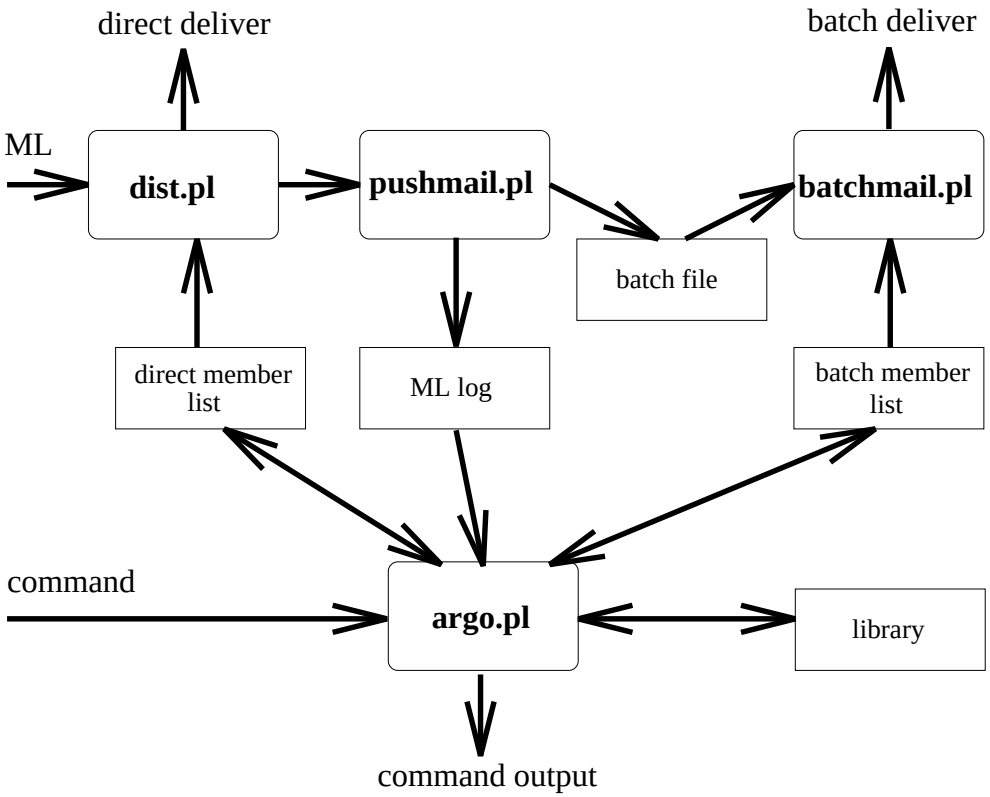

図 1: ARGO システムのデータの流れ

[2] K. Moore, “MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text,” RFC1522, 1993.

[3] Simson Garfinkel, “PGP: Pretty Good Privacy,” O’Reilly & Associates, Inc., 1994.

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

楯岡 孝道 (tree@mma.club.uec.ac.jp)

Abstract

MMA が利用しているプライベートネットワークでは、学外とのメール交換に何らかの中継機構が必要である。MMA では、ネットワーク接続開始当初は静的配送を用いて中継を行っていたが、後に DNS の MX レコードを利用した中継機構を開発し、利用している。本文章ではこの機構について述べる。

1 はじめに

電気通信大学では、一部のサブネットワークに RFC1597 [4] で規定されているプライベートネットワークアドレスを用いて、ネットワークアドレスの有効利用を行っている [1]。

MMA は 1995 年 4 月から学内ネットワークに IP 接続を行っているが、MMA の接続しているサブネットワークも、このプライベートネットワークアドレスを用いている。

しかし、プライベートネットワークアドレスを用いているホストは学外との IP による直接の通信が行えないため、通常用いられる DNS の MX レコードを用いたメール配送 [3] を行う事ができない。

本文章では、本学で利用されているプライベートネットワーク関連の機構と、直接の対外接続をもたないホストにおけるメール配送の問題点を述べた後に、MMA が利用してるメール配送機構について述べる。

2 プライベートネットワーク

プライベートネットワークとは、RFC1597 によって割り当てられた組織内のみで利用可能な IP ネットワークアドレスを利用したネットワークである。

現在、世界的に IP ネットワークアドレスが枯渇し、このままでは数年後には新たな割当てができなくなると言われている。一方で、組織内には直接組織外と IP 接続を行わなくてもよいホストが存在する。しかし、これらのホストに正式に割り当てられていない IP アドレスを利用すると、組織内の直接対外接続を行っているホストから該当する IP アドレスを正式に割り当てられたホストへの到達性が失われてしまう。そのため、従来はこのようなホストに対しても新たに正式な IP アドレスを取得してきた。

RFC1597 ではある範囲の IP アドレスに関して特定の組織への割当を行わない事を宣言している。これによって安全にこのアドレスを組織内のみ有効なアドレスとして利用できるようになった。

当然ながら制限もあり、該当アドレスに関する情報は組織外に流してはならないとされている。すなわち、経路情報や DNS のレコードなどに該当アドレスに関する情報が入ってはならないという事である。

特に DNS[5] に関しては、組織内のホストからは該当アドレスに関する情報も取得する必要があるため、工夫する必要がある。詳細は前回の百万石の記事 [1] に譲るが、本学ではプライベートネットワークのホストは通常のホスト名に “.private” をつけた名前で DNS に登録する事により、これに対処している。例えば、DNS のデータベース上では MMA のホスト

```
beat.mma.club.uec.ac.jp
```

は、

```
beat.mma.club.uec.ac.jp.private
```

として登録されている。

この特殊なホスト名は正式なホスト名でないために学外から参照される事は無い。よって、このようなレコードを DNS に登録しても RFC1597 には違反しない。

3　通常のメール配送方法

通常 Internet 上でのメール配送は、DNS の MX レコードにしたがって行われる [3]。同一メールアドレスに対して MX レコードは複数存在して良く、複数の経路を用意する事により対故障性を高めている。

しかし、組織外と直接通信できないホストが組織外とメールを交換するには、何らかの中継機構を用意する必要がある。特に組織外からのメールをどのように処理するかが問題になる。

この問題はプライベートネットワークアドレスを使わない場合も生じるので、いくつかの一般的な解決方法が存在する。

3.1　静的配送

DNS の MX レコードには中継ホストを登録しておき、中継ホストに静的中継ルールを用意する事により、配送を行う方法である。DNS 普及前は全て静的配送を行っていた事もあり、広く利用されている。

この場合、静的配送ルールの維持管理と耐故障性が問題になる。

ネットワークの接続状況は日々変化しており、静的配送ルールの維持を困難にしている。特に MMA のようにサブドメイン上のホストが全てプライベートネットワーク上に存在する場合は、中継ホストを公共ホストに頼らざるを得ず、ルールの維持は各方面に多大な負担を強いることになる。

対故障性に関しては、複数の中継ホストを用意する事により解決できるが、その場合は静的配送ルールの維持が更に困難になる。

3.2　重みつき MX による配送

DNS の MX レコードには、優先度の数字をつける事ができる。これを利用して中継を行う事ができる。

メール配送では最初に優先度の高いホストに対する接続に失敗すると、次に高いホストに対して配送を試みる。したがって、低い優先度で中継ホストを登録しておくと、組織外のホストは目的のホストに接続しようとして失敗し、続けて中継ホストにメールを配送する。中継ホストでは目的のホストに接続できるので、正しくメールが配送される事になる。

この方式の利点は、メール配送に関する情報を DNS の MX レコードで集中管理できる事と、中継ホストには特別の設定が必要無い事である。

欠点としては、組織外ホストが目的ホストに到達できない事を送信ホストに伝える事ができない場合、送信ホストは SMTP[2] 接続のタイムアウトまで待たないと、中継ホストへの接続を行わない事があげられる。

また、プライベートネットワークの場合、DNS を用いて組織外に IP アドレスを伝える事ができないので、この方法は利用できない。

無理に .private を付加した不正なホスト名を MX レコードに登録すれば、学外からはその IP アドレスが引けず、結果として中継ホストに到達しそうだが、MX に登録されたホストの IP アドレスが引けないと即配送エラーにしてしまう sendmail の実装が存在するので、この方法も現実的ではない。

3.3　3 系統 DNS の利用

DNS を、組織内のみ参照/組織内外を参照/外部からの参照 の 3 系統にわけて運用している場合、それぞれに異なる MX レコードを設定する事により対処できる。

すなわち、外部からの参照用の DNS には MX レコードに中継ホストを登録し、組織内の DNS には目的のホストを登録すれば良い。

この方法は、DNS の維持管理が複雑になる以外の弊害はないが、本学では 1 系統の DNS のみで運用しているため、この方法は利用できない。

4　MMA における配送

MMA ではネットワーク接続当初は、静的配送によってメールの中継を行い、後に DNS の MX レコードを利用した配送を行うようになった。

4.1　静的配送

静的配送では、本学の公共サービスマシンである `ns.uec.ac.jp` において `club.uec.ac.jp` 及び `*.club.uec.ac.jp` 宛のメールは全て MMA のサーバホスト `beat` に配送する静的配送ルール

を記述してもらい、beat において更に細かい静的配送を行う事により配送を行っていた。DNS の MX レコードとしては club.uec.ac.jp 及び *.club.uec.ac.jp 宛メールを ns に配送するように登録されており[1]、全てのホストからのメールは一度 ns と beat を経由して配送された (図 1)。

この配送方式には、先に述べた静的配送における 2 つの問題が存在した。すなわち配送ルールの維持管理と対故障性である。

ルールの維持管理に関しては、一度全ての club サブドメイン宛メールが beat まで配送されるので、サークル内の beat 側で club ドメイン内部の配送ルールを記述でき、ある程度緩和されたが、ホストが増えるだけで設定を変更する必要があるため、繁雑であった。

一方、サブドメインの配送ルールを beat 側に持ったため、2 つの中継ホストを経由してメールが配送される事になり、耐故障性は低くなった。つまり、ns か beat のどちらかでトラブルがあれば club ドメイン宛のメールは全て停止するという状況であった。

4.2 PRIVATE HACK による配送

MMA では 8 月末から、広く利用されている MTA (Mail Transfer Agent) である sendmail [6][7] の設定ファイルの書き換えにより、DNS の MX レコードを利用した配送に切替えた。ここではこの手法を PRIVATE HACK と呼ぶ。

この方式では、sendmail による配送において、次の規則を付加する事により配送を実現している。

1. 配送先メールアドレスに .private を付加した IP アドレスが存在するかどうかを検査する
2. 存在した場合、.private を付加したメールアドレスを用いて MX レコードの検索を行い、配送先を決定する
3. 存在しない場合、通常通りにそのまま MX レコードの検索を行い、配送先を決定する

配送には少なくとも中継ホストでこの規則を付加する必要がある。MMA の場合、ns.uec.ac.jp でこの規則を付加してもらっている。

現在では、DNS に次のような MX レコードが登録されている[2]。なお、優先度は小さい方が優先される。

さらに、メールアドレスに .private を付加したアドレスには適当な IP アドレスを登録し、IP アドレスが引けるようにしている。ほとんどのメールアドレスはホスト名と一致するので、新たに必要な必要なアドレスはわずかであった。

この変更を行った事により、従来は beat を経由して配送されていた fan 等の下位ドメイン宛のメールも ns から直接それらのメールアドレスのサーバに配送されるようになり、対故障性と配送速度が向上した (図 2)。また、MX を複数指定する事により、目的ホストにトラブルが生じても、中継ホストの ns にメールが溜る事を防ぐ事も可能になった。

この方式の利点は、大きくわけて 3 つあげられる。

1. 一度中継ホストにルールを付加すれば、プライベートアドレスを用いたドメインが増加しても中継ホストの設定変更は必要無い
2. メール配送に関する設定が DNS により一元管理できる
3. 中継ホスト以外にこのルールを付加しても通常の配送に影響せず、付加したホストからプライベートアドレスを用いたドメインへの配送が直接行われるようになる

特に 3 つめの特徴により、この付加を行っても特別な管理を必要とせず、中継ホストを複数用意することが容易になっている。

5 実装

PRIVATE HACK の実装は、最初 sendmail の設定ファイルの sendmail.cf の書き換えにより実現され、つぎに sendmail.cf 生成ツールである CF の localhack ルールとして記述された。

sendmail.cf の書き換えでは実質的に 6 行、CF では冗長な処理を省略する事により 2 行で記述さ

[1] 実際のレコードでは ns の別名である tweedledee.cc.uec.ac.jp が登録されている

[2] もちろんこれは一部である

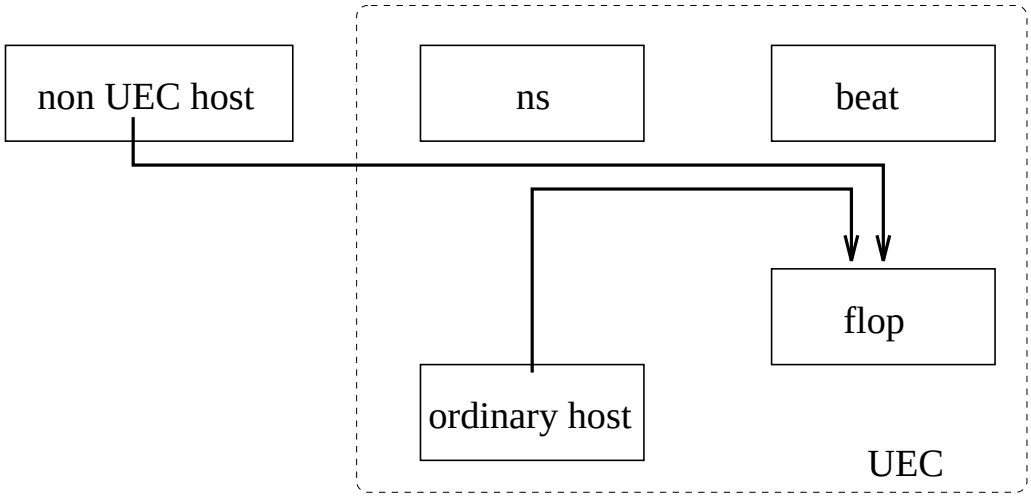

図 1: 静的配送による配送経路

メールアドレス	優先度	MX ホスト名
club.uec.ac.jp	100	ns.uec.ac.jp
.club.uec.ac.jp	100	ns.uec.ac.jp
club.uec.ac.jp.private	0	beat.mma.club.uec.ac.jp.private
mma.club.uec.ac.jp.private	0	beat.mma.club.uec.ac.jp.private
fan.club.uec.ac.jp.private	0	flop.mma.club.uec.ac.jp.private
fan.club.uec.ac.jp.private	10	beat.mma.club.uec.ac.jp.private

れている。CF への書き換えは本学計算機センターの小林克志先生の手による。

6 問題点

PRIVATE HACK の問題点として、存在しない private ドメインへのメールが中継ホストに到着した場合、local configuration error として処理される点があげられる。このエラーは、そのドメインの postmaster に通知されるのが望ましいが、現在はそのような機構は用意されておらず、中継ホストの postmster に通知される。

7 評価

現在、MX レコードによるプライベートネットワークへの配送機構の付加は、中継ホストの `ns.uec.ac.jp` と、一般ホストの `twoheaded.cs.uec.ac.jp` で行われ、2ヵ月以上運用を行っている。現在 MMA には学外から週に約 600 通強のメールが到着しているが、運用中 sendmail.cf の入れ換え時のミスによるトラブルが一度あったのみで、順調に動作している。

配送経路に関しては、中継ホストが一台減った事によりメール配送がよりスムーズになった。

また、今回の sendmail の設定により、今後プライベートネットワークのみのサブドメインが新たに出現しても、DNS への登録のみでメール配送を受ける事が可能になった。また、サブドメインでサブドメイン内のプライベートネットワークへの中継を行う際にも、そのままで利用可能である。

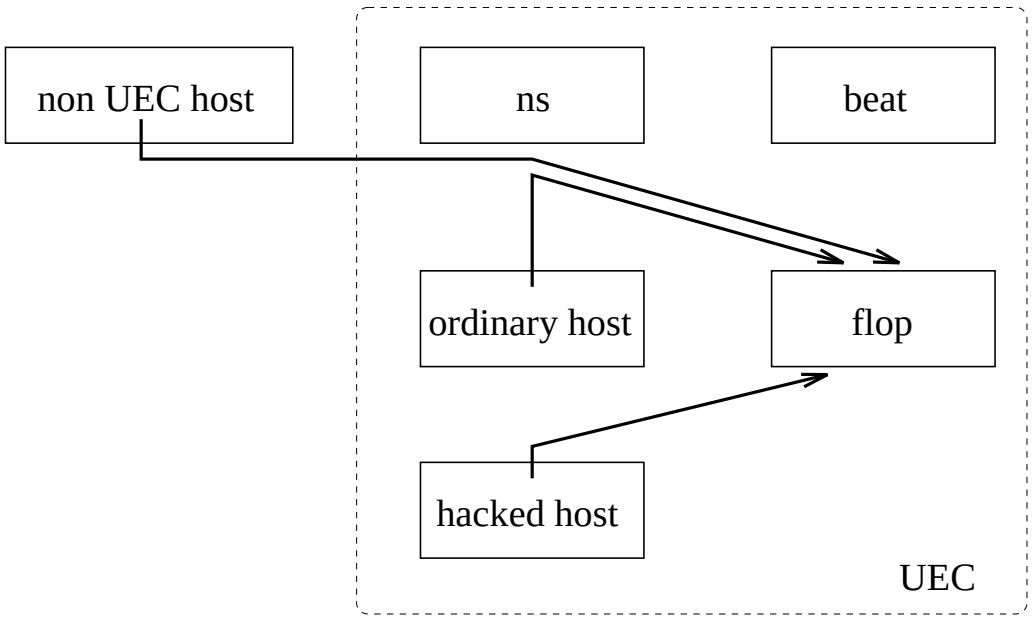

図 2: PRIVATE HACK による配送経路

8 まとめ

PRIVATE HACK を利用する事により、プライベートネットワークに対しても効率が良く信頼性の高いメール配送機構を構築する事ができた。またメール配送経路の管理も DNS で一元管理でき、維持管理が容易になった。

今後さらなる中継ホストが確保できれば、複数の中継ホストを MX レコードに登録する事により、さらに対故障性が確保できると期待できる。

謝辞

PRIVATE HACK のアイデアを提供していただき、さらに CF に移植する事により可搬性を高め、繰り返してのメールホストの設定変更にも快く応じてくださった本学計算機センターの小林克志先生に心より感謝します。

参考文献

[1] 楯岡 孝道, “Private Network Address に於ける DNS 環境の構築,” MMA 会誌 百万石, pp.18-pp.20, 電気通信大学 MMA, Apr.1995.
also available from URL:
`http://raven.cc.uec.ac.jp/home/edu/a-94e/i9410011/http/MMA/h_9504_dns.html`

[2] J. Postel, “Simple Mail Transfer Protocol,” RFC821, 1982.

[3] C. Partridge, “Mail routing and the domain system,” RFC974, 1986.

[4] Y. Rekhter, R. Moskowitz, D. Karrenberg, G. de Groot, “Address Allocation for Private Internets,” RFC1597, 1994.

[5] M. Lottor, “DOMAIN ADMINISTRATORS OPERATIONS GUIDE,” RFC1033, 1987.

[6] Eric P. Allman, “Sendmail Installation and Operation Guide, Version 8.36,” SMM:08-1, 1994.

[7] Bryan Costales, with Eric Allman and Neil Rickert, “Sendmail,” A nutshell handbook, Sebastopol, CA : O’Reilly & Associates, 1994.
邦訳: 鈴木 克彦訳, “Sendmail 解説,” インター

ナショナル・トムソン・パブリッシング・ジャパン, 1994.

不定期連載 ハードボイルドとコンピュータ
第一回「依頼人がほしい」

4C boss

1　前置き

ハードボイルド[3]の主人公は、コンピュータの事をどう思っているのだろうか。

ふとそんなことが気になって、読了して段ボールにしまった本の読み直しをしてみた。

だけではつまらないので、簡単にコメントをつけてみる事にした。

2　本題

終了キーを押すと、文書を「保存しますか」ときいてきた。イエスをあらわす “Y” を打って実行キーを押した。

すると、画面が真っ白になったのだ。

パッと消えた。なくなったのだ。

(略)

どうあがいても、取り返しがつかない。

主人公は作家を目指す中年[4]の私立探偵、という名の事故専門の調査員。拳銃もコンピュータもまったくわからない。

そんな彼が妻に押されて、はじめて PC で原稿を書くことになった。その結果が冒頭の引用である。もちろん、「保存」したのだから、呼び出す事は容易に出来る。だから、「なくなった」のは、勘違いである。

だが、パニックに陥った頭にはそんな考えは出てこない。単に操作を知らなかっただけだが、これ以後、主人公はコンピュータ恐怖症になってしまう。「いまいましい機械に」「こちらの神経がまいってしまった」のだ。

こんな機械などいじくれなくても、頭さえあれば探偵業は務まるよ、といったところであろうか。

自動車と同様、コンピュータもユーザがある程度、仕組みを知らない[5] といけないと思うのだが、不思議とそれを言う人は少ない。

かくいう筆者も、研究室の先生にそれを言う勇気は無かったりするのだが。これ以上書くと身の危険を感じる[6]ので、ここらで退散する事にしよう。

パーネル・ホール 田中一江訳
「依頼人がほしい」
ハヤカワ・ミステリ文庫
1993 年 12 月 31 日発行
1994 年 2 月 28 日二刷
定価 640 円

[3]ハードボイルドとはなんぞや、という問いに、答えは無い。本連載では筆者が (これはハードボイルドだ) と思ったものをハードボイルドとする。

[4]少年老い易く、学なり難し?

[5]ハードディスクにショックを与えてはいけない etc.

[6]そう、研究室で書いているのだ。

へなちょこ三次元ポリゴンプログラム

uirou@mma.club.uec.ac.jp

昔、僕が高校生のころ、PC98 とかいうコンピュータの上で動く「MIND」というコンパイラで、リアルタイムでポリゴンを描くというプログラムを作った事があった。しかし、「MIND」は計算が多少遅く、計算速度命のリアルタイム処理では涙の出るような速度でしか動かなかった。

で、今回、UNIX 上で C とかいう言語を使って同じようなものを書いてみた。実は、よくみてみると完全にできているわけではないのだが、基本的な所は一応できている (はずな) のでこのまま記事にしてしまおうと思う。

僕のやろうとしたことは、(偉そうにいえば) 三次元でポリゴンを描くだけでなく、ポリゴンで物体を定義し、それが関節をもっているようなデータ形式で作りたかったのである。とりあえずどんなことかというと、まぁ、ゲームセンターや、よく言う次世代機とかで動いている、格闘ゲームって感じだ。あれっていうのは、(たぶん、僕の思うに) 体の中心から、手足が生えてて、体の中心を回転させると、手足はそれに応じて動くというものなはずだ[7]。そこで、関節おきに物体を定義して、その物体どうしをディレクトリツリーのような形でつなげて考えるということにした。この考え方でいくと、例えば、手を表すのに “/胴体/肩についている所/肘についている所/手” となる。ここで、注意するのが、「手」の所のデータがいくら回転しても、「胴体」には影響を及ぼさないが、「胴体」を回転させると、そのしたの「肩についている所」も、「肘についている所」も、「手」も全て影響を受け、「肘についている所」を回転させると、「手」の所には影響を与えるが、「肩についている所」より上の所には影響は無いということだ。

ということで、とりあえずデータ形式を作ってみた。まぁ、こんな感じだ [例 1]。

ここで、

- `*p[]` が、一枚づつのポリゴン
- `m` が、その物体の回転率 (3x3 の行列だ)
- `maisuu` が、その物体のポリゴンの枚数
- `*next_object[]` が、その物体の下にある物体 (ディレクトリが掘ってあるわけだ)
- `mannaka` が、その物体の中心位置[8]

である。

これを上から順番に計算すればいいと思ったのだが、世の中そう簡単にはいかなかった[9]。上から順番に計算したのはいいものの、それをそのまま順番に書いたのでは順番がめちゃくちゃで表示がおかしくなってしまった。要するに、画面からみて向こうの方にあるものから順番に描いていかなければならないのに、そんなことお構い無しに描いてしまったのである。そのため、一度計算したデータを蓄えておいてそれを Z 軸でソートするというプログラムを書いた。

そこまでできて、とりあえず動いたので、とりあえず MMA のロゴを入力してみた。もう知っているだろうが、MMA のロゴはこんな感じである [図 1]。

この M と、A を作って入力し、真中の M の左右に M と A を配置したのだが、このまま表示したら、ジグザグの所がむちゃくちゃになってしまった。ジグザグの所の隠面処理をしていないからである[10]。そこで、M と A を 4 つと 3 つに分割したデータにした [図 2]。

これで (物体間の前後関係は見ているので)、きちんと表示されるようになった[11]。

これでさぁできたと思って喜び勇んで学校に持って行き、実行したら、「ちらついてるねー」と言われてしまった[12]。「がーん！しし、しまったぁ！」家

[7] でも、もしかして、計算結果だけを連続して表示してるだけだったりしたら、やだなぁ。

[8] 僕のものぐさで maisuu なんてのがあるけど、普通はそんなの無いんだろうなぁ...

[9] いつもの事なんだけどね。

[10] 当然、そんなことは考えていなかった...実は今でもしていない。

[11] そのせいで無駄計算が増えたんだけど。

[12] 関係のない話だが、そう言われた時に、自分の家のマシンとの表示速度の違いに愕然としていたという話がある。ちな

```
struct OBJECT {
polygon *p[MAX_POLYGON_MAISUU];
_3dmatrix m;
unsigned int maisuu;
struct OBJECT *next_object[MAX_NEXT_OBJECT];
_3dpoints mannaka;
};
```

例 1: データ形式

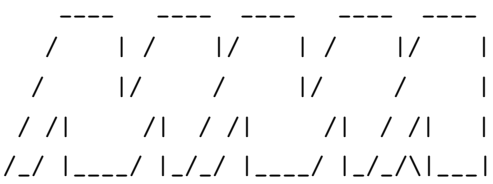

図 1: MMA のロゴ

では、「ちょっとちらついてるかなー」とは思っていたのだが、実際に言われてみるとけっこうこたえるものがあった[13]。そこで、「一度グラフィック画面でないところに描いて、それをコピーすればいい」ということに気づいて、部室にある X の本を読んだ。都合良く "`XCopyArea()`" なんてものがあったので、それを使う事にした。(幸い) ちょっといじっただけでそこのところは変更する事ができた。

しかし！これだけでもいけなかったのである[14]。画面からみて向こう側から描くのをチェックするところでは、Z 軸の所しかみていなかった。そのため、Z 軸が同じ場所のものは順番がどうなるか解らなかったのである。「どうせそんなの、気にせぇへんでえーやろ」と思って、放っておいたのだが、横一列に並んだものを横から順番に描いて行くと、どうしても見えてはいけない「横」の部分が見えてしまった[15]。これを直すには、多分、X の自乗と Y の自乗を加えたものが大きい方から順番に描いて行けばいいはずだ[16]。

まぁ、これでプログラムは完成したわけだが、僕には絵のセンスというものが余り無いので[17]MMA のロゴをどういう風に動かそうか考えていなかった。そのため、ただグルグル回るだけになってしまったのである。[18]

最後に、とりあえず僕がこのプログラムを組んだ時に役に立ったと思った本を列挙しておく [表 1]。お役に立てば幸いである。

みに、僕の家のマシンはペンちゃん 100 で、学校のマシンは 486-66 である... ビデオカードが違うんだけどね...

[13]家の X がビデオカードに対応してなくて、そのせいで遅いんだけどね...

[14]やっぱり、いつもの事なんだけどね

[15]当然なんだけどね。

[16]でも、これって、まだ、直してないんだよね。

[17]全然無いとは、思いたくない

[18]もし、時間があったら、直してあるはずだ... 多分そんな時間無いけど。

図 2: 改良した MMA のロゴ

UNIX C プログラミング	アスキー
入門 実習 応用グラフィクス	アスキー
UNIX X ウィンドウプログラミング (Xlib)	HBJ 出版局

表 1: 役に立った本

perl なスカラデータのひみつ

しみずりょう (仮名)

1 INTRO

perl には 3 つのデータ構造があります。それらは、スカラ (Scalar Variable = SV)、配列 (Array Variable = AV)、連想配列 (Hash Variable = HV) と呼ばれます。配列や連想配列は、スカラを格納するためのものなので、結局、スカラを扱うことになります。

スカラ変数とは、なんでも格納することのできる変数、と言ってよいでしょう。perl の変数は、あいまいな awk のそれや、すべてを自分で管理しなければいけない C に比べて、扱うのがとても楽です。具体的にどう楽なのかというと、

- 数値、文字列を意識しなくても扱える
- 意識すれば、数値は数値、文字列は文字列として扱え、変換が容易
- 文字列には、7bit, 8bit, コントロール文字, null を問わず、なんでも使うことができる
- 文字列の長さに制限がない

などです。

1.1 数値、文字列を意識しなくても扱える

これは、いちいち itoa() などのような関数を使わなくても、“100” という文字列の入ったデータを、数値として評価すれば 100 という数値が得られる、ということです (これは AWK といっしょ)。

1.2 意識すれば、数値は数値、文字列は文字列として扱え、変換が容易

ある関数には、数値ではなく文字列を渡したい、あるいはその逆、という状況が発生する場合もあります (引数が数値な場合と文字列な場合で動作が異なる関数、等)。

文字列を数値にするには、普通 0 を足すことによって変換します。すなわち、perl は、式のコンテクストによって、評価値を数値、文字列に型を変えてくれます [例 1]。

1.3 文字列には、7bit, 8bit, コントロール文字, nul を問わず、なんでも使うことができる

文字列は char の配列と見なして、なんでも入れることが可能です。長さは perl が管理しているので自動的に増減してくれます。

```
$str = "\x00\x01\x02\x03";
```

は、長さ 4 の char の配列といった具合です。

とうぜん、変数に (テキスト、バイナリ問わず) ファイルの一部を読み込む、などといったこともできます。

```
open(FILE,"/vmunix");
sysread(FILE,$data,1024);
```

とすると、$data には、/vmunix の先頭 1024 バイトが格納されることになります。

バイナリファイルにパッチを当てたい、という場合。

例えば、”foobar” という名前のファイル (ファイルサイズ 12345) のオフセット 678 バイト目のデータを、0x4e にしたい、といったような場合なら、これだけで済みます [例 2]。

AWK ではこんな低レベルなことまでできませんし、C だともうちょっと行数が必要となるでしょうし、コンパイルもしなければなりません。

1.4 文字列の長さに制限がない

さらに、文字列の長さは、perl が厳密に管理しているので、AWK のような、処理系によって使える文字列の長さが違う、といったようなあいまいな制限がありません。

文字列は、メモリが許すだけ使えます。やろうと思えば、10 メガバイトの文字列だって使うことができます。

```
$mma = "100";           # $mma は文字列の "100"
$mma += 0;              # $mma は数値の 100
$mma .= "";             # $mma は文字列の "100"
                        # ( . は文字列連結演算子 )
```

例 1: 数値と文字列の変換

```
open(FILE,"+>foobar");          # 読み書きで open
sysread(FILE,$buff,12345);      # 12345 バイト read
substr($buff,678,1) = "\x4e";   # 678 文字目を 0x4e に
                                # substr は左辺値としても機能する!
syswrite(FILE,$buff,12345);     # 12345 バイト write
close(FILE);                    # close
```

例 2: バイナリファイルへのパッチ当て

文字列が長くなれば perl が内部でメモリを確保してくれるし、短くなれば効率よくメモリを開放してくれます。

2 SCALAR VALUE(SV)

perl の sv (Scalar Value) の、C での実装レベルでいろいろ調べてみることにしましょう。

sv 構造体は、以下に示されるとおりです [例 3]。(perl のソースキットに含まれる、sv.h と sv.c でスカラ変数の処理を行っています)

sv_flags には、そのスカラの中身の種類が格納されます。以下のような感じで定義されています [例 4]。

その SV が空ならば SVt_NULL, int 値なら SVt_IV、double 値なら SVt_NV、リファレンスなら SVt_RV、文字列なら SVt_PV ... といった具合です。SVt_PV[IN]V については後述します。sv_flags が SVt_NULL でない場合は、sv_any に、その内容物 (または内容物の構造体) へのポインタが入ります。sv_refcnt は、sv の参照箇所数です。リンクカウンタと言った方が適当かもしれません。参照 (ハードリファレンス) される度にインクリメントされます。参照が減り、この値が 0 にると、sv 構造体が開放されます。

あるスカラ変数に、123 という数値を格納する場合、内部では以下のような動作をしています [例 5]。

浮動小数点型に関しても、int が double に変わるだけです。実際には、もうちょっと複雑な処理をしている訳ですが、やっていることはおおよそこんな具合です。

文字列の場合はちょっとややこしくなります。単純に考えると、sv 構造体の sv_any に、malloc した文字列へのポインタを格納すればうまくいくような気もしますが、それだと文字列が NULL で終わることが前提となってしまします。文字列の場合はさらに構造体を定義し、あらゆる文字列 (ひいては、あらゆるバイナリデータ) が扱えるように工夫されています [例 6]。

このような構造体へのポインタを、sv->sv_any に格納します。すなわち、文字列そのものは、sv->sv_any->xpv_pv と 2 つのポインタを辿った所に格納されることになります。ちなみに、xpv_cur は評価される文字列のサイズ、xpv_len は、文字列の為に malloc されているサイズです。2 つのサイズがあるのは、文字列を切り詰めたり広げたりする際に、不必要に malloc/realloc をするのを防ぐためです。よって、xpv_cur が xpv_len より大きくなることはありません。

あるスカラ変数に、
“WeAreMMA\0WeAreNotHackers” という文字列を格納するには、

```
typedef int             I32;
typedef unsigned int    U32;

struct sv {
    void*    sv_any;      /* pointer to something */
    U32      sv_refcnt;   /* how many references to us */
    U32      sv_flags;    /* what we are */
};
```

例 3: sv 構造体

```
enum {
        SVt_NULL,       /* 0 */
        SVt_IV,         /* 1 */
        SVt_NV,         /* 2 */
        SVt_RV,         /* 3 */
        SVt_PV,         /* 4 */
        SVt_PVIV,       /* 5 */
        SVt_PVNV,       /* 6 */
          :
```

例 4: スカラの中身

```
struct sv *my_sv;
my_sv = malloc(sizeof(sv));          /* 新規に SV 構造体を確保 */
my_sv->sv_refcnt = 1;                /* 参照数は 1 */
my_sv->sv_flags = SVt_IV;            /* int 型 */

my_sv->sv_any = malloc(sizeof(int)); /* 内容物用 */
*(my_sv->sv_any) = 123;              /* ここに内容が入る */
```

例 5: スカラ変数の内部動作

```
#define size_t  unsigned int  /* 処理系による */
#define STRLEN  size_t

struct xpv {
    char *      xpv_pv;      /* pointer to malloced string */
    STRLEN      xpv_cur;     /* length of xpv_pv as a C string */
    STRLEN      xpv_len;     /* allocated size */
};
```

例 6: 文字列の場合

```
struct sv *my_sv;
struct xpv *tmp_xpv;

my_sv = malloc(sizeof(sv));     /* 新規に SV 構造体を確保 */
my_sv->sv_refcnt = 1;           /* 参照数は 1 */
my_sv->sv_flags = SVt_PV;       /* 文字列型 */

{
        int len;
        len = sizeof("WeAreMMA\0WeAreNotHackers");

        /* xpv 構造体を確保 */
        tmp_xpv = malloc(sizeof(struct xpv));

        tmp_xpv->xpv_pv = malloc(len);
        strcpy( tmp_xpv->xpv_pv,
                 "WeAreMMA\0WeAreNotHackers");
        tmp_xpv->xpv_cur = len;
        tmp_xpv->xpv_len = len;
}

my_sv->sv_any = tmp_xpv;
```

例 7: スカラ変数への文字列の格納

おおよそこんな具合です [例 7]。文字列には NULL も含まれる可能性もあるため、strlen などの関数を使うべきでないところがポイントです。

さて、次はこれらの型変換についてです。なにも考えずに実装するならば、SV の型を相互変換するには、その都度、それ用の型の構造体を作成、型変換、格納、もとの型の構造体を開放... といった手順を繰り返す、で終わりなのですが、さすがは perl というかなんというか、型変換に要する実行速度を少しでも速くするためにいろいろな工夫がなされています。

その前に、SVt_PVIV、SVt_PVNV という型について説明をしておきます。SVt_PV[IN]V という型は、文字列と整数、あるいは、文字列と整数と実数を同時に持つ型です。構造体を以下に示します [例 8]。

これらの構造体は、SV の型変換の操作を容易かつ高速にするためにあります。例えば文字列 (SVt_PV) 型を整数 (SVt_IV) 型に変換する際には、実際には SVt_IV ではなく、SVt_PVIV 型に変換します。SVt_PVIV 型は、文字列のフィールドも持っているので、SVt_PVIV 型から SVt_PV 型への変換は特殊な場合を除いて行う必要はありません。さらに、SVt_PVNV 型は、文字列と整数と実数を持っている型で、この型に行き着くと、他のどの型への変換も行わずに済みます。

当然、これらの型は、真に有効な型が何であるかもちゃんと保持されています。sv->sv_flags に、真の型が何であるかのビットがあり、それを見ることで有効な型が判ります。

... と、perl の SV についてほんの一部説明しましたが、実際は、さらに複雑な処理をしています。perl はとてもよく考えられて作られてますので、機会があれば一度ソースを読んでみることをお勧めします。

```
  struct xpviv {
    char *  xpv_pv;    /* pointer to malloced string */
    STRLEN  xpv_cur;   /* length of xpv_pv as a C string */
    STRLEN  xpv_len;   /* allocated size */
    IV      xiv_iv;    /* integer value or pv offset */
  };

struct xpvnv {
    char *  xpv_pv;    /* pointer to malloced string */
    STRLEN  xpv_cur;   /* length of xpv_pv as a C string */
    STRLEN  xpv_len;   /* allocated size */
    IV      xiv_iv;    /* integer value or pv offset */
    double  xnv_nv;    /* numeric value, if any */
  };
```

例 8: 文字列と数字を同時に持つ型

百万石用スタイルファイルの作成

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

Abstract

MMA は、毎年新歓の時期と調布祭の時期に、「百万石」という同人誌を出すことが慣例となっている。

今回は、佃編集長の意向により、LaTeX による、ほぼ完全な電子化を行うということになった。そこで、私がスタイルファイルの面倒を見ることとなった。

1 スタイルファイルの仕様

1.1 要求された仕様

今回の百万石の編集責任者である、佃さんの要求は、一言でいえば「UNIX MAGAZINE のようにしてくれ。」ということでした。

具体的には

- 図表はページの上の部分に入れる。
  `figure[t]` 環境で実現可能である。スタイルファイルに手を加えて、`figure` 環境のデフォルトの位置を設定しなおすべきである。
- 注釈は、ページの下の部分に入れる。
  `footnote` コマンドで実現可能。これは、各人が投稿に `footnote` コマンドを用いるようにする。
- 二段組みにする。
  `jtwocolumn` スタイルで実現可能。ルートファイル[19]に記述する。

ということが挙げられます。

1.2 細かな仕様

佃さんの要求仕様は、見た目に関することのみでした。その他にも、LaTeX で多人数の投稿を処理するのですから、ルートファイルと `input` コマンドを使用した処理をできるようにしたいです。

[19] 文書処理システム LaTeX 75 ページ

そのために、具体的には

- 各人の論文は、必ず奇数ページの先頭から始まること。
- 各人の論文の始めで、各種カウンタが初期化されること。
- 論文のヘッダー (氏名、題名等) の書式が統一されること。
- 原則として、`jarticle` スタイルの書式でソースファイルを書くことができること

といった機能を実現する必要があります。

2 スタイルファイルに組み込む機能

百万石スタイルファイルには、次の機能を組み込みます。

- `figure` 環境の領域のデフォルトをページ上部にする。
- 各人の論文の区切りを示し、カウンタの初期化や、ヘッダーを作成するコマンド、`article` を作る。その書式は、`\article{題名}{著者}`とする。
- 上下左右のマージンや行間隔等の調整をする。ただし、この機能は百万石用スタイルファイルではなく、`b5j.sty` 等のファイルに手を加えるべき物である。

3 スタイルファイルの作成

スタイルファイルは、`jarticle.sty` をベースに、細かな修正を施して作成した。これにより、「`jarticle` スタイルの書式でソースファイルを書けること」という要求はほぼ満たされるはずである。

4　今後の課題

- 百万石スタイルファイルをオプションスタイルファイルにする。
  こうした方が汎用性や拡張性、保守性が高まるからである。
- ヘッダやフッタの設定をもう少しマシなものにする。
- 目次の自動作成機能を実現する。
- abstract を二段抜きで組版する。

などが挙げられる。他にも、細かい調整箇所はたくさんありそうだ。次回の百万石までにはスタイルファイルを完成させたい[20]。

また、overfull \hbox を出にくくするための方法についても、何か考えたい。

5　最後に

最終的な百万石用スタイルファイルは、ここに書かれた物よりも機能が拡張されていると思う。この文章と、スタイルファイルの作成は並行して行われ、締切が近づけば、スタイルファイルの作成の方を優先するからである。

最終的な成果は、次の百万石に投稿する予定である[21]。

[20] 余談だが、私は「未完成のプログラムを実戦に投入する」ことは得意である。高校時代の駅伝大会成績集計プログラムも...

[21] ちょっと今回は忙しかった。

‘login:’ までの道

uncover@mma.club.uec.ac.jp

皆さんは普段使っている UNIX システムが、どのように起動するかを知っていますか? ここでは UNIX の中でも BSD 系と呼ばれるシステム、そしてその中でも、NetBSD/i386 という OS に対象を絞って、‘login prompt’ が出て来るまでを、ざっと見ていきます。NetBSD/i386 というのは 386 クラス以上の CPU を持つ、AT 互換機で動作する OS で、そのため AT 依存の部分も多少ありますが、大筋の流れにおいて、他の BSD システムも似たりよったりです。

Step 1: 火を入れよう

計算機の電源を入れると、harddisk の MaterBootRecord(MBR) と呼ばれるセクタを、BIOS がメモリに読み込み、そこへ実行を移します。MBR から読み込まれたプログラムは、中に組み込まれたパーティション情報をもとに、ブートすべきパーティションを決定し、アクティブパーティションのブートセクタをロードし、実行を移します。こうしてアクティブパーティションに存在する OS が実行されるわけです。NetBSD では、xxboot という実行イメージが MBR にロードされ、次に bootxx が xxboot によってロードされます。そして最後に、NetBSD のカーネルイメージ (例えば ‘/netbsd’) が bootxx によって選択され、NetBSD が起動します。

Step 2: /netbsd が起動したら

カーネルと呼ばれるプログラムは、UNIX システムの核をなす部分で、NetBSD では /netbsd にその実行イメージが格納されています。カーネルは起動時に、プロセス環境、メモリ、I/O デバイスなどを初期化しなくてはなりません。なによりもまずカーネルは、コンソールを初期化します。コンソールが初期化されないと、画面表示を行う事ができないので、とても不便です。次に仮想記憶を初期化します。これで、メモリ保護を行う事ができるようになりました。いよいよ次にデバイスドライバの初期化を行います。カーネルは、デバイスが存在するかどうかをテストするために、デバイスドライバの中に probe と呼ばれる関数のエントリをもっており、それぞれのデバイスに対応する、probe エントリを呼び出す事によって、そのデバイスが存在するかどうかを確認します。probe エントリによってデバイスの存在が確認された場合、カーネルは次に、そのデバイスを実際に初期化するために、attach と呼ばれるデバイスドライバのエントリを呼び出します。attach エントリでは、デバイス固有のデータ領域の初期化や、割り込みハンドラの設定、物理的なデバイスの初期化を行います。

以上で Hardware の初期化は大体終了したので、プロセスが実行できるように、Software の初期化を始めます。これらにはプロセス構造体の初期化、ファイル記述子テーブルの初期化、ファイルシステムの初期化などが含まれます。カーネルはこの段階で、0、1、2、三つのプロセスを生成します。プロセス 0 は swapper と呼ばれ、scheduling をします。プロセス 1 は /sbin/init を実行し、全てのユーザプロセスはこのプロセスの子どもとなります。プロセス 2 は pagedaemon と呼ばれ、ページングを行います。

こうしてカーネルは三つのプロセスに分かれ、これから後は init だけが子プロセスを生成していくことができます。

Step 3: /sbin/init が走る

init プロセスは /etc/rc を実行します。rc は通常 sh-script で、機種依存度の低いシステムの初期化を行います。例えば、ファイルシステムのチェックや、ホスト名の設定、ネットワークインタフェースの初期化、標準的なサービスを行うプロセスの生成などです。この rc が正常に終了した場合、いよいよ init は /etc/ttys を読み込んで、login を受け付けるべき端末に、しかるべきプロセスを張

り付けます。

Step 4: getty が見張る

/etc/ttys には端末に関する情報が記述されています。その中には init がその端末に張り付けるべきプロセスが書かれており、通常は getty が指定されています。getty は端末の回線を見張り、回線速度や端末の種類など、回線特有のさまざまな設定を終えた後、login プロセスを実行します。

Step 5: login prompt

login プロセスはその名のとおり、端末からのログインを受け付けるプロセスで、こいつが login prompt を表示し、ユーザの入力を待ちます。私達はこのプロセスに対して、ユーザ名とパスワードを入力し、ログインすることができるわけです。

こうしてとうとう、ユーザがログインできるところまで、システムが起動しました。これらの流れはおおまかなものであり、簡単のために省略された部分もかなりありますし、ネットワークを通じてのログインでは、かなり違う部分もあります。また X–Window を用いる場合では、xdm と呼ばれるプロセスが login プロセスの仕事を肩代りします。しかし、BSD のシステムが起動する際のメッセージを眺めながら、“ああ、今このあたりの処理をしているんだなぁ” くらいのイメージは持てるのではないでしょうか?

1995年 調布祭 百万石

編集責任者　　佃 良生
表紙デザイン　　宮宗 正樹
編集協力, 組版　君島 秀征

1995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@uma.club.uec.ac.jp) までお願いします。