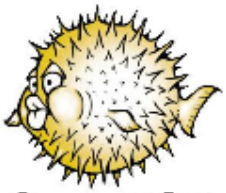
百萬石

MMA '07年 秋号



パイレーツすらも
食い殺すワイルドさ!

知ってたか？　孔雀は
堕天使の象徴なんだぜ？

LaTeX

エレガントに舞い、
タイプセットに酔う

ガイアが俺にもっと
輝けと囁いている

漆黒に選ばれし男の
体制への逆襲

1つだけ言える真理がある。
「男は黒に染まれ」

この迷彩がヤバ過ぎる
牙を程よく包んでくれる

ここからが俺の
伊達ワルレジェンド
のはじまり

いつだって何かに
逆らい生きてきた

目次

第1章 otenkiプロジェクトの経過と現状

(著者) 奥村ゆうき
mjt@cltn.org

概要
2004 年度から、MMA では情報基盤センター (総合情報処理センター： 2004 年当時) に設置されているインターネット百葉箱に相当するシステムを内製するプロジェクトに取り組んでいる。
本稿では、その経過と現状のプロジェクト概要について説明する。

1 インターネット百葉箱の概要

1.1 製品構成

情報基盤センターのシステムは、DAVIS 社の GroWeather 気象センサと Linux-PC を組み合わせたインターネット百葉箱とよばれる製品で構成されている。気象センサは総合研究棟の屋上に設置されており、Linux-PC は最上階の 10 階に設置されている。

PC にはデータベースと Web サーバが内蔵されており、ページの生成やデータの蓄積を行っている。

1.2 採用しているセンサと提供されている情報

GroWeather は、風速計、太陽放射強度、温度、湿度、紫外線強度、雨量、カメラの各センサを備える。
また、PC が生成する Web ページ (http://weather.cc.uec.ac.jp) によって、それらのログやグラフが提供されている。

2 試作物

幾つかのシステムを試作した。
以下、制作順に列挙する。

2.1 第 1 次サービスサーバ (2004 年調布祭)

PostgreSQL によって温度データを記録し、HTTP の cgi でグラフを生成して表示させる最初のサーバを制作した。

- シリアルポートからの SHT71(温度/湿度データ) の読み取り
- PostgreSQL によるデータの記録
- cgi によるログの参照

を実装していたが、

- 更新処理が遅い
- グラフの生成が非常に遅く、デフォルトの自動更新にも対応できない

という問題があった。

2.2 第 1 次風速/風向センサ (2005 年調布祭)

PSoC を利用して、風速/風向センサの読み取りを試行した。

風速/風向センサは、プロペラの回転による内蔵された発電機の出力の波形を読み取ることで風向を知り、本体の回転を内蔵されたポテンショメータの抵抗値の変化で知る仕組みとなっている。

そのため、適当なスレッショルドによって波形を 2 値化し、それをカウントすることでプロペラの回転数を知るという手順を踏む必要があった。

- PSoC 内蔵 A/D による風向センサ波形/抵抗値読み取り
- シリアルポートからのデータ出力

を実装していた。

2.3 第 1 次観測システム (2005 年,2006 年調布祭)

秋月電子の H8-LAN ボードに TOPPERS/JSP 1.3 と TINET を利用した観測システムを試作した。
デジタル/アナログセンサを IPv4 化するというコンセプトで制作された。

- syslog プロトコルによる温度/湿度/雨量/風向/気圧センサ状態の通知
- tftp プロトコルによるリモートからのファームウェア更新

を実装していたが、

- 不明な原因で不定期に停止する

という問題が有ったため採用は見送られた。

2.4 第 2 次観測システム (niji：開発中止)

第 1 次システムの動作停止の原因を TOPPERS に求めて、独自のカーネルを開発することで問題の解決を図った物。設計等の詳細は別稿を参照の事。

DRAM を搭載したボードを屋外に設置すること自体がそもそも安定性を欠くなどの意見により、この方向性は廃止され、以降の観測システムはより小規模なマイコンに依る物を指向することとした。

2.5 第 2 次サービスサーバ (2006 年調布祭)

PostgreSQL の代わりに、コンピュータネットワークの観測ログ記録で定評のある RRDtools を採用することで問題の解決を図った物。

- RRDtools による観測システムからのデータ (syslog) の記録
- RRDtools によるグラフ化

を実装していたが、

- 過去のログを参照できない
- グラフの生成処理はやっぱり重い

などの問題がある。

3 現在の計画

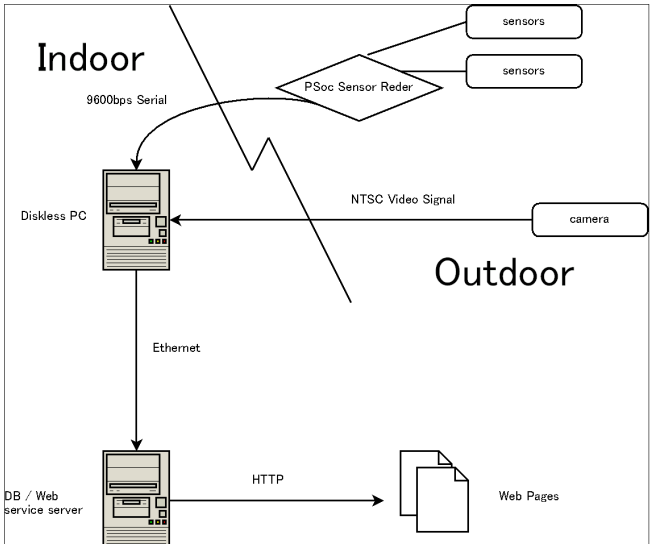


図 1 otenki system

現在は次のような構成を想定して実装を行っている。
プロジェクトは 2009 年 3 月の完了を目標に作業を行っている。

3.1 PSoC による Sensor reader

PSoC を利用して、各種センサをシリアルで読み取れるように変換する。
風向/風速計以外のセンサは単純な A/D や独自のシリアルプロトコルで動作するため、比較的容易に実装できる物と考えられる。
PSoC 側の実装はほぼ終了し、今後は PC 側のクライアントソフトウェアを開発することとなる。

3.2 Diskless PC による現地 PC

ディスクレス PC によって、PSoC から受信した観測データと、カメラの NTSC ビデオデータをキャプチャし、Ethernet で DB/Web サーバに送信する。
以上の PSoC と Diskless PC は観測地の現地に設置することを前提としている。

3.3 DB/Web service server

DB/Web service server は現地 PC からのデータを受け取って DB(観測データ) またはファイル (画像データ) に格納し、Web ページや観測データのグラフを生成する。
このサーバは電通大に設置する予定となっている。

第2章 niji : The Internet Friendly O/S without MMU

(著者) 奥村ゆうき
mjt@cltn.org

概要
niji[*1]は、インターネットフレンドリーを特徴とする組み込み向けの省メモリシステムを目指して開発されているオペレーティングシステムで、プロトコル記述とロジックを分離することによって高い移植性を得ることを目標としている。
本稿では、カーネルと設計手法について概説する。

1 ネットワーク化された省メモリ OS の必要性

近年、いわゆる組み込み機器の高度化により、Linux 等の PC や WS 向けの OS が、組み込み機器においても搭載されるケースが増えてきた。

しかし、それらの OS は MMU を前提としてデザインされているため、MMU が無い状況では uClinux 程度の選択肢しか無い。そのため、多くのネットワークを扱う組み込み機器は MMU を搭載した CPU と数 MB 以上の RAM を搭載するようにデザインされる。

そこまでして組み込み UNIX を採用するのは、

- 豊富なプロトコル実装
- 動作実績
- 安定性

等が求められていることに依る。そこで、本研究では「豊富なプロトコル実装」を free-standing 環境 (OS や既存のライブラリの支援の無い環境) で実現することを可能な限り簡単に実現する方法について検討し、それらを纏める最低限のカーネルデザインを目指した。

2 プロトコル記述言語 RAIN

本研究では、プロトコルの記述を独立した言語で行い、その言語を RAIN と呼ぶ。

RAIN は、XSLT や Lisp に影響を受けた宣言型の言語で、実際のプロトコル実装はコンパイラが必要なパートを推論して C 言語に変換することによって行う。

2.1 ソースファイルの形式

例えば、RAIN で記述された PING に応答するだけの ICMP は以下のようになる。実際には [Echo Request] を受信するタスクが別に必要となる。

```
*[ICMP Packet] |seq| [TLV Frame]
        [Type] : [byte] [TLV Type]
        [Code] : [byte]
        [Checksum] : [word] |target| [ICMP Packet] |algo| [inet]
        [Identifier] : [word]
        [Seqence number] : [word]
        [Data] [TLV Value]
--

*[Echo Reply] [ICMP Packet]/0
        value : [byte] [string]
```

[*1] niji と RAIN は同名の異なるアプローチの実装を毎年行っている。本稿で述べる niji と RAIN は、2006 年度から今年に掛けて otenki プロジェクトのためにデザインされた物

```
--

*[Echo Request] [ICMP Packet]/8
        value : [byte] [string]
--

*[ICMP] [module]
        [IP Protocol Type] : 1
        [root] : [ICMP Packet]
--
```

他のプログラミング言語と大きく異なるのは、シンボルに空白を含んだ文字列をよく使う点と、プロトコル実装でよく利用される語彙を予め備えているという点にある。

RAIN のソースコードは、

- 構造体の定義 (順序付き/順序無し)
- 条件式の定義

を行う。定数は静的な条件式として定義される。

2.2 RAIN のコード生成

RAIN のコンパイラは C 言語のソースコードを出力する。

構造体のアクセスは、C 言語の構造体は利用せず、常に unsigned char などの C 内蔵型の配列を利用したコードにコンパイルされる。

構造体の要素サイズはビット単位で管理されているため、フラグやニブルなどの要素が構造体中に含まれるプロトコルであっても問題なくコンパイルすることが出来る[*2]。

要素の配置は適切な優先度を持って行われる。すなわち、構造体のアラインはパディングを後で配置することによって実現されている。また、ビット単位の配置も、bit packing rule を外部で記述することによってかなり自由に設定することができ、プロトコルのリトルエンディアンやビッグエンディアンの違いもこの段階で吸収している。

ただし、現在の RAIN コンパイラはリトルエンディアン向けのコードしか出力しない。

TLV 構造を持つプロトコルや長さフィールドに応じて長さの変わる構造体は、静的なコードとしてアクセスコードを生成できないため、次に説明する動作推論によってコードが生成される。

2.3 RAIN における動作推論

RAIN を他のプログラミング言語と異なる物にしているのは、現実的に利用される戦略についてあらかじめ記述された静的なコード (= テンプレート) を自動的に付与する前提で動作推論が行われる点にある。

現在、RAIN は

- 可変長データの処理
- TLV 構造のマッチと構成
- CRC やチェックサムの検証
- クエリ結果のキャッシュ

を必要に応じて推論し、生成したコードのドライバやスタブとして付与する。

RAIN はメモリ保護を持たない niji を前提にデザインされているため、この動作推論によるコード生成は、セキュリティを高める手段としても利用されている。例えば、いわゆる TIFF 脆弱性の多くは TLV のパースに起因するが、RAIN は予め丹念に書かれた TLV 実装を様々なプロトコルで流用することによって、プロトコルの独自実装に起因する問題を回避している。

キャッシュなどを推論によって生成するようにデザインすることで、異なる規模の実装であっても同じプロトコル記述から生成することが可能になる。組み込みプロセッサでは、アドレス空間によってバス幅が異なるなどの事情があるため、一つのオブ

[*2] はずだが、今のところかなりバグが有る

ジェクトの情報を分散して格納する必要が考えられる。そのような時に、推論によるテンプレート適用は、オブジェクトの分散を自動化することが出来る。

3 niji カーネル

3.1 カーネルの記述

カーネルにおいて記述すべき C のコードは殆ど無いため、システムの移植性を向上させるのに役立っている。
必要な関数は、

- 起動時の初期化やスタックの設定
- コンテキストの切り替え
- 割り込みの通知
- イベントキューのための安全な mutex

程度で、また、実装すべきリソースのサイズは実装者が自由に選択して RAIN にフィットを任せることが想定されている。ただし、現状では、コンテキストは割り込みの数 +1、スタックサイズは 8KB に固定されている。

カーネル側にはバッファの管理などの機能は要求されておらず、もしもメモリが必要な場合は静的に領域を確保する必要がある。

RAIN の生成するコードを単なるライブラリとして用い、カーネルとして TOPPERS/JSP 等の既存のリアルタイムカーネルを用いるという方向性も当然考えられる。その場合、niji 本来のカーネルは必要としない。

3.2 イベントシステム

niji のカーネルはイベントドリブンの疑似マルチタスクカーネルとなっている。通常、niji のターゲットとする領域では RTOS を搭載することが多いが、niji はシステムのイベントを細分化することで人間の体感レスポンスタイムには影響が無い程度の応答性能を目指す。

プロトコルの処理は中断可能なものと中断不能なものに予め分類されており、RAIN 記述を行う際には中断不能期間が 100ms を超えないように配慮する必要がある。常識的なプロトコルの処理にはそれほど時間は掛からないのでこの決めうちによる問題は発覚していないが、信号処理のような複雑なタスクを記述した場合には問題になるかも知れない。

現状では殆どのプロトコルはプロトコルの処理が終わるまでは中断不能として記述されている。つまり、ペイロードをパースしてメモリ上の構造体に書き込むまでは一切の中断は許可されない。ただし、パースした構造体を上位プロトコルに渡す段階で再度イベントキューを経由するため、それなりの粒度でマルチタスク性が保たれている。

niji のタスクは一般的な OS と異なり、システムを構成 (= カーネルをコンパイル) した段階で全てが決定されている。このデザインは uITRON のような RTOS のデザインを踏襲している。つまり、組み込み機器のタスクは大きく変化しないというのが前提となっている。この仮定によって、カーネルには静的にイベントハンドラのアドレスが書き込まれることとなり、システムの ROM 化を容易にしている。

niji のイベントは全て RAIN で記述されたシンボルによって識別される。例えば [ECHO Request] がイベントとして発生し、もしも、予め [ECHO Request] に対応するタスクがカーネルに定義されていればそれが実行される。

RAIN の生成するイベントは複数の詳細度を持つため、例えば [ECHO Request] は同時に [ICMP Packet] であり [IP Packet] であり [802.3 Packet] である。それらについても、処理するタスクが定義されていれば処理される。

これらのディスパッチは RAIN 記述によって生成され、カーネル内に静的にコンパイルされる。

3.3 objs protocol

RAIN で定義するイベントを直接配信するプロトコルとして objs protocol を定義している。これは 1280octet 以上のパケットサイズを持つパケット型の通信路を想定してデザインされていて、インターネット経由ではポート 6809 で動作する UDP アプリケーションとなっている。

これは、CORBA 等の ORB に相当するプロトコルと言えるが、objs のプロトコル自体も RAIN によって生成されるため、機

器間での互換性は自動的に確保されるわけではない。そのかわり ORB 等に比べて実装を簡単にし、必要なリソースを減じることが出来る。

4 実装

現在は Linux/NetBSD 向けのエミュレータを実装している。

RAIN によって、TCP/IPv4、UDP、DNS、ARP、ICMP(Echo request の応答のみ)、IEEE802.11 のインフラストラクチャモード、zd1211b 向けの NIC デバイスドライバ、USB 1.1 デバイスプロトコルが実装されているが、RAIN コンパイラ側の問題で正常に動作しない部分が多い。

現状での消費リソースは 40KB 程度であり、より省メモリ化を進めていくことで CPU 内蔵 RAM でも動作可能にすることを目標としている。

5 これからのトピック

5.1 niji を利用した CMS の制作

現在、niji を使用した CMS の制作に取り組んでいる。これは、文章の投稿を RAIN のイベントとして扱い、それを記録/変換することで HTML ページを作る物で、著者の日記 (diary.okotama.org) として試作中のものが動作している。

将来的には osdev-j の Wiki(wiki.osdev.info) で運用する予定となっている。

objs protocol を遅延させるためのファイルシステムを制作する必要がある。前述の通り objs protocol は通信する機器間でプロトコルが異なる可能性があるため、(パフォーマンスを犠牲にして) 標準化したプロトコルを設計することを考えている。

5.2 遅延名前解決

現在、RAIN の名前解決はカーネルのコンパイル時に一括して行われるが、これを実運用時まで遅延することでローダブルモジュールをサポートしたいと考えている。

これにはセキュリティ上の問題があるため、何らかの形で形式的にプロトコル記述を検証する手段が必要となる。

5.3 オープンソースソフトウェアとしての公開

現在でも、AVE-TCP のような組み込み向けネットワークスタックは多く販売されているが、FOSS においてはあまり有力な実装が存在しないため、niji+RAIN でそのようなセグメントにおいても FOSS の存在感を与えることができればと考えている。

RAIN はプロトコルの実装に特化したデザインとなっているため、C 言語よりも簡単に典型的なプロトコルを実装することが出来る。この特性を生かして、「プロトコルの Wikipedia」と呼べるようなリポジトリを作っていきたい。

第3章 他力本願で目立とう

(著者) 奥村ゆうき
mjt@cltn.org

概要
なんか今年は他力本願な行為でよく目立った年だった気がする。

1 YouTube で目立とう

Youtube に MMA の鍵システムの動画[*3]を投稿したらかなり激しいアクセスを記録した。

図 2 部室ドア前でなぜか wii リモコン振る俺

1.1 システムの概要

MMA の鍵システムに関しては何度か記事にしているので、過去の百萬石[*4]を参照してもらうとして、今回のシステムは、いわゆる wii リモコンを振ると鍵が開いたりしまったりする機能拡張をしました。

Bluetooth は FreeBSD の実装を使って、HID パケットのパースは手で書いてます。

1.2 FreeBSD からの Bluetooth

ぶっちゃけ bluetooth(3) の man を見てもらえば一瞬で終わりだと思いますが、なんと普通の Socket に見えます。まぁWinCE の IrSock とかでもおなじみの形式なので特に驚くような事では無いと思いますが。

```
int
mksocket(int psm){
        struct sockaddr_l2cap addr;
        int s;
        s = socket(PF_BLUETOOTH,SOCK_SEQPACKET,BLUETOOTH_PROTO_L2CAP);
        addr.l2cap_family = AF_BLUETOOTH;
        addr.l2cap_psm = 0;
        str2ba(LADDR,&addr.l2cap_bdaddr);
        if(bind(s,(struct sockaddr*)&addr,sizeof(addr))<0){
                printf("error in bind()\n");
```

[*3] http://jp.youtube.com/watch?v=ksi4jm_Gkjk
[*4] http://delegate.uec.ac.jp:8081/club/mma/mma/hyakuman.xhtml

```
        }
        memset(&addr,0,sizeof(addr));
        addr.l2cap_family = AF_BLUETOOTH;
        addr.l2cap_psm = htole16(psm);
        str2ba(RADDR,&addr.l2cap_bdaddr);
        if(connect(s,(struct sockaddr*)&addr,sizeof(addr))<0){
                printf("error in connect() (%d) \n",errno);
        }
        return s;
}
```

LADDR がローカルの BT アダプタの MAC アドレス、RADDR が制御対象の MAC アドレスになります。

TCP 的なポート番号の代わりに psm という番号があります。デバイスのコントロールと、実際のデータ読み取りで別々の番号が割り当たってるのが普通で、Wii リモコンの場合は 0x11 と 0x13 でした。

後は霊感でコントロールして[*5]、加速度計のデータだけ拾えればあとは鍵を開け閉めするライブラリとリンクするだけ。簡単。一日で出来てたと思います。

1.3 動画の撮影と編集

動画の撮影には普通のデジカメの動画撮影機能を使ったと思います。一応、見栄えの良い wii リモコンの振り方をかなり試行錯誤したので、撮影には 2 時間くらい掛けたと思います。

編集は Windows XP 上の Windows Movie Maker を使って、gimp で作った文字部分と足しています。ニコニコ動画とかそれ系のサービスと違って youtube では自分で頑張ってエンコードする余地が無いので、ビットレート高めの wmv に書き出してアップロード。

1.4 掲載後の反応

最初のうちはあんまり反応が無かったんですが、本家 Engadget に例の微妙にバカにされた口調で取り上げられて (Wiimote hacked to make opening doors look silly) からは一日に 4 万再生とかかなり意味不明なトラフィックを記録し、ちょっとそれどうなのという気分になりました。

その後、gizmodo とか kotaku のようなメジャーな blog になぜか次々登場し、最終的には 16 万再生まで行ってしまいました。

ちなみに、日本語圏の blog で取り上げてくれたところは殆ど無かったです。中国語圏では結構人気になったっぽいですが、良くわかりません。

1.5 効果と教訓

もともと、新入生勧誘の意図で作ったんですが、その年の一年生はゼロという大変男らしい結果に終わったので、「Youtube で人気になっても新入生は来ない」という結論が出たと言うところです。

流行物のデバイスを使うと有名になれるというのが教訓。当時は wii リモコンの活用が熱かった時期の末期で、次は wii fit の板辺りで遊ぶと良いかも知れません。

研究を見せるツールとしての youtube は結構有効な手段なんじゃないかという手応えは有りました。

2 MonaOS のインタビュー受けてきた

いわゆる@ IT で MonaOS の開発者にインタビューというので、開発者の端くれとして行ってきました。結果のインタビューは「開発者が語る Mona のこれから[*6]」として載ってます。

[*5] 実際には wiili.org 辺りで解析資料を探した方が楽だと思います

[*6] http://jibun.atmarkit.co.jp/lcom01/rensai/comtan03/comtan01.html

2.1 ページに載らなかったこと

そもそも、これってキャリアとかそれ系のスペースに載ってる気がするのに、インタビューを受けた当人達は無職 (ひげぽん氏:当時)、学生 (僕) とそれはどうなのというメンツ。。

実際には、ひげぽん氏は在職中にどうやって仕事と Mona を両立したのかみたいな話を聞かれていたんですが、なぜか記事中には出てないです。そして僕にはその手の話は一切無かったのがなんとも言えない。まぁ、そんな事聞かれても話すこと無いんですが。

基本的にインタビューはひげぽん氏に質問をして僕が必要に応じて言葉を補うというスタイルでやっていました。

ちなみに、実際の記事として@ IT に載る前に、我々には最終稿をレビューする権限が与えられています。というかそれが当たり前だと思ってたんですが、そんなの無しにいきなり載るという激しいメディアも有るそうで、インタビューを受けるときはそういうのに気をつけた方が良いと思います。どうやって気をつけるのかはちょっと分かんないですが。

2.2 後悔したことと教訓

活字になる時のことを考えて発言しないといけないというのが教訓。

実際に文字になった記事を見ると解ると思いますが、僕の発言が全体的に何様だお前という感じであんまり良い印象を受けないですね。。実は、レビューの段で結構文末を弄ってこれでも印象が柔らかくなるように変更してます。実際の発言はもっとアレだったということで。

予習をしていかなかったのも良くなかった。実はどんな質問があるのかみたいな事項は先にメールで伝えられていたのに、それに対してちゃんとした答えを考えないまま当日を迎えてしまったのが良くない。

あと、写真を撮らなかった！こういう記事を書くときに写真が無いと説得力が無い。。

それから、所属がインタビューに載ってないのも誰なんだお前という感じでよろしくない。これからは電通大 MMA とかを prefix に付けようと思いました。

3 おわりに

どうも目立つことはよろしくないみたいなイメージがやっぱり自分の中にあって、ちょっと前の自分だったら絶対にインタビューとかは受けてないと思います。

ただ、やっぱり開発者本人が体を張らないとプロダクトは浮かばれないんじゃないかと思って、去年辺りから雑誌記事[*7]書いたり、積極的に表に出るようにしました。まだその効果が現れるような時期では無いと思う[*8]んですがこれからも体を張っていこうかなと考えています。

[*7] 今は亡きオープンソースマガジンに「デバイスとしての FeliCa と PC 連携」って記事を書きました

[*8] 実際、仕事のオファーをくれる人の大半は web 日記とかのオンライン活動から僕の事を知ってくれるようなので

第4章 IPsec で簡単 IP 暗号化通信

(著者) 電気通信大学電気通信学部情報通信工学科 谷島英典
yajima@mma.club.uec.ac.jp

1 IPsec とは

IPsec とは、IP パケットの改竄防止、さらに暗号化を提供するプロトコルである。 RFC により標準化されており、*BSD, GNU/Linux, Mac OS X, Windows2000/XP/Vista 等、各種 OS で標準で利用可能である。

IPsec は複数のモードや、様々なアルゴリズムを利用可能であり柔軟性に富んでいるが、それゆえ複雑になり接続性に問題が起きてしまい、一般的に利用が進んでいるとは言えない状況である。

そこで本稿では、IPsec をより多くの人に使ってもらうべく解説する。簡単に利用するため、IPsec トランスポートモード, ESP に内容を絞って説明を行うことにした。

2 実験環境

2 台の FreeBSD ホスト (rise, nagasaki) 間を IPsec トランスポートモードで接続し、暗号化通信を行った。

- rise (FreeBSD 6.2-RELEASE)
- nagasaki (FreeBSD 7.0-BETA3)
- Racoon2(racoon2-20070720a)

3 設定

鍵交換デーモンである、Racoon2 の設定をするだけで利用可能である。 OS 等の設定は不要なので、非常に簡単に IPsec を利用できる。

設定は/usr/local/etc 以下の、下記のファイルを適切に設定するだけである。

- racoon2.conf
- vals.conf
- transport_ike.conf

3.1 racoon2.conf

```
include "/usr/local/etc/vals.conf";

# interface info
interface
{
        ike {
                MY_IP port 500;
## Uncomment to enable NAT-T (both initiator and responder)
#               MY_IP port 4500;
        };
        spmd {
                unix "/var/run/racoon2/spmif";
        };
        spmd_password "/usr/local/etc/spmd.pwd";
};

# resolver info
resolver
{
        resolver off;
```

```
};

include "/usr/local/etc/default.conf";

## Transport mode IKEv2 or IKEv1
include "/usr/local/etc/transport_ike.conf";
```

sample をほとんど書き換える必要はない。
複数の IP アドレスがあるホストの場合なら、MY_IP に直接 IP アドレスを書くらいである。
それと、今回はトランスポートモードを用いるので、それ以外をコメントアウトしただけである。

3.2 vals.conf

```
setval {
### Directory Settings ###
        # Preshared key file directory : specify if you want to use preshared keys
        PSKDIR          "/usr/local/etc/psk";

        # Cert file directory : specify if you want to use certs
        CERTDIR         "/usr/local/etc/cert";

### ID Settings ###
        # your FQDN : specify if you want to use FQDN as your ID
        MY_FQDN         "rise";

        # Peer's FQDN : specify if you want to use FQDN as peer's ID
        PEERS_FQDN      "nagasaki";

### Preshared Key Setting ###
        # Preshared Key file name
        # You can generate it by pskgen.
        PRESHRD_KEY     "test.psk";

### Certicate Setting ###
        # Set following parameters if you use certificates in IKE negotiation
        # and _SET_ 'kmp_auth_method { rsasig; };' in each remote{} section of
        # tunnel_ike{_natt}.conf/transport_ike.conf files.
        # For more information, please see USAGE.
        #
        # Your Public Key file name
        MY_PUB_KEY      "my_pub.pem";

        # Your Private Key file name
        MY_PRI_KEY      "my_pri.pem";

        # Peer's Public Key file name
        PEERS_PUB_KEY   "peers_pub.pem";

### Transport Mode Settings ###
        # Your IP Address
        MY_IPADDRESS    "10.0.139.6";

        # Peer's IP Address
        PEERS_IPADDRESS "10.0.60.66";
```

このファイルで、通信するホストに関する情報を設定する。必須なのは、MY_FQDN, PEERS_FQDN,MY_IPADDRESS, PEERS_IPADDRESS である。それぞれ、自分と相手の IP アドレスを設定する。

nagasaki 側では それぞれ逆に設定する。

鍵交換に用いる方法は、PRESHRD_KEY と x509 証明書が利用可能である。今回は簡単に済ますために、preshared key を用いた。安全性を考えるならば、証明書を用いるべきであるが、現在の racoon2 の実装では証明書の自動交換ができないようなので、ランダムで長い preshared key なら十分であろう。

それぞれのホストで /usr/local/etc/psk/test.psk に preshared key を設定する。

3.3 transport_ike.conf

```
remote ike_trans_remote {
        acceptable_kmp { ikev2; };
        ikev2 {
                my_id fqdn "${MY_FQDN}";
                peers_id fqdn "${PEERS_FQDN}";
                peers_ipaddr "${PEERS_IPADDRESS}" port 500;
                ## Use Preshared Key
                kmp_auth_method { psk; };
                pre_shared_key "${PSKDIR}/${PRESHRD_KEY}";
                ## Use Certificate
                #kmp_auth_method { rsasig; };
                #my_public_key x509pem "${CERTDIR}/${MY_PUB_KEY}" "${CERTDIR}/${MY_PRI_KEY}";
                #peers_public_key x509pem "${CERTDIR}/${PEERS_PUB_KEY}" "";
        };
        selector_index ike_trans_sel_in;
};

selector ike_tarns_sel_out {
        direction outbound;
        src "${MY_IPADDRESS}";
        dst "${PEERS_IPADDRESS}";
        upper_layer_protocol "any";
        policy_index ike_trans_policy;
};

selector ike_trans_sel_in {
        direction inbound;
        dst "${MY_IPADDRESS}";
        src "${PEERS_IPADDRESS}";
        upper_layer_protocol "any";
        policy_index ike_trans_policy;
};

policy ike_trans_policy {
        action auto_ipsec;
        remote_index ike_trans_remote;
        ipsec_mode transport;
        ipsec_index { ipsec_esp; };
        ipsec_level require;
};
```

ここでは、transport mode の設定を行う。 preshared key を使うか x509 証明書を使うかの設定をする。sample では psk を使うような設定なのでそのままの状態にしておく。

残りは、upper_layer_protocol の設定である。ここでは、IPsec を利用する上位プロトコルを指定する。今回はすべての通信を暗号化するので、‘‘any’’ とした。

設定は以上で完了である。

4 起動方法

4.1 セキュリティポリシ管理デーモンの起動

最初に以下のコマンドで、セキュリティポリシ管理デーモン (spmd) を起動する。SP(セキュリティポリシー) とは、どのホストとの通信を IPsec で行うかを決定するものである。

```
/usr/local/etc/rc.d/spmd.sh
```

SPD(セキュリティポリシデータベース) の確認は以下のコマンドで行う。

```
 # setkey -DP
10.0.60.66[any] 10.0.139.6[any] any
        in ipsec
        esp/transport//require
```

```
        created: Oct 19 13:24:04 2007  lastused: Nov 22 17:50:26 2007
        lifetime: 0(s) validtime: 0(s)
        spid=16437 seq=1 pid=40360
        refcnt=1
10.0.139.6[any] 10.0.60.66[any] any
        out ipsec
        esp/transport//require
        created: Oct 19 13:24:04 2007  lastused: Nov 22 17:50:26 2007
        lifetime: 0(s) validtime: 0(s)
        spid=16436 seq=0 pid=40360
        refcnt=1
```

以上のように表示されれば正常にできている状態である。

4.2 IKE デーモンの起動

次のコマンドで IKE デーモンを起動する。IKE とは鍵の自動交換を行うものである。IPsec では安全性を高めるため、定期的に (時間、通信量が一定値を超えたら) 鍵を更新する。

```
/usr/local/etc/rc.d/iked.sh
```

これで IPsec による通信ができるようになっているはずである。 IPsec で通信ができていれば、次のコマンドで SAD (セキュリティアソシエーションデータベース) を確認できる。SA とは、暗号化方式や暗号鍵等にかんする情報である。

```
# setkey -DP
10.0.60.66[any] 10.0.139.6[any] any
        in ipsec
        esp/transport//require
        created: Oct 19 13:24:04 2007  lastused: Nov 22 17:50:26 2007
        lifetime: 0(s) validtime: 0(s)
        spid=16437 seq=1 pid=40360
        refcnt=1
10.0.139.6[any] 10.0.60.66[any] any
        out ipsec
        esp/transport//require
        created: Oct 19 13:24:04 2007  lastused: Nov 22 17:50:26 2007
        lifetime: 0(s) validtime: 0(s)
        spid=16436 seq=0 pid=40360
        refcnt=1
```

4.3 接続が切れた場合の対処方法

ホストが再起動したなど、接続が切れた場合、IPsec の通信も利用できなくなることがある。これは、鍵の有効期限が切れるまで通信不能になるためである。この場合、SAD を以下のコマンドで消去することにより初期化させることで対処する。

```
#setkey -F
```

5 VPN としての利用

IPsec トランスポートモードにおいては、暗号化される IP パケットはホスト間のみであり、セキュリティゲートウェイとしては利用できない。そこで、この通信路上に IP over IP トンネルや、L2 Bridge を構築することにより、拠点間の暗号化通信を可能にする。 L2 bridge として動作させれば、同じセグメントの LAN 接続している状態にみせかけることができる。これは、電気通信大学助教である楯岡先生作による gluebridge(`http://www.tateoka.org/%7Etate/diary/20030209e/index.html#gluebridge`) などを利用すると非常に簡単に実現可能である。 本稿では、FreeBSD の IP over IP トンネルの実装である、gif(4) を用いて VPN を構築方法を記述する。

5.1 実験用ネットワーク構成

rise, nagasaki を VPN ゲートウェイとして利用する。外側の IP アドレスは以下とする。

```
rise     10.0.139.6
nagasaki 10.0.60.66
```

内側のネットワークは以下とする。

```
rise     192.168.0.254/24
nagasaki 192.168.6.1/24
```

5.2 設定

rise, nagasaki それぞれのホストで、以下のように gif トンネルを作成する。

rise

```
ifconfig gif0 create
ifconfig gif0 192.168.0.254 192.168.6.1 tunnel 10.0.139.6 10.0.60.66
```

nagasaki

```
ifconfig gif0 create
ifconfig gif0 192.168.6.1 192.168.0.254 tunnel 10.0.66.66 10.0.139.6
```

これでトンネルは設定できたので、残りはルーティングテーブルを書けば、両ネットワーク間で通信可能となる。

rise

```
route add -net 192.168.6 192.168.6.1
```

nagasaki

```
route add -net 192.168.0 192.168.0.254
```

6 まとめ

以上のように Racoon2 を用いれば、非常に簡単に IPsec による暗号化通信が利用可能になる。VPN を利用したいと考えているのならば、怪しげな独自プロトコルの VPN など利用せずに、IPsec を積極的に活用して欲しい。

7 今後の課題

現状では IPsec 通信を行うには、それぞれに global なアドレスが必要であり、NAT 以下の環境では利用できない。しかし、UDP-ESP トンネルによる NAT Traversal によりそれが解決できそうなので、その実験を行う予定である。

また、FreeBSD 6 の FAST_IPSEC(IPv4 only) および FreeBSD 7.0 以降の IPsec では、ハードウェア暗号化アクセラレータを利用可能である。今回は、vpn1401 を用意したが、認識はするものの、crypto device として利用できなかった。

今後は、これらのアクセラレータを利用して、それにともなうスループットの増加および CPU 負荷の軽減を計測する予定である。

第5章 シェルスクリプトによるloggingの改良

(著者) moechar
moechar@mma.club.uec.ac.jp

概要

どうも，本誌編集の moechar です．本稿では春号で紹介した logging 方法の改良版を記述しています．なので，高品位なスクリプト言語を喋れたり UNIX 系 OS に造詣が深い読者の皆さんには極めて無用なので早急に読み飛ばすことをオススメします．

1 はじめに

春号で懸念していた IED の計算機環境全体にわたるリプレースの件ですが，ホスト名とホスト数が変わっただけで特に変わりはなかったようです.[*9]しかし `ied{1...4}`と `master` 以外のホストでは相変わらず `default shell` の `tcsh` に権限が与えられていないためまともに活動できないのはどうにかして欲しいところです。。。今回は，

- 春号で示したスクリプトの最適化及び可読性の向上
- `load average` を記録するスクリプトの作成 (`recloadaverage.sh`)
- ログイン状況を多様的に把握できるように `recw.sh` の改良

の 3 点を行いました．

2 改良後の logging system

`load average` を記録する機能が実装されたのでこの様になっています．

(1) log file の格納場所の設置 (`makedir.sh`)
(2) ログイン log file を作成 (`recw.sh`)
(3) `load average` の log file の作成 (`recloadaverage.sh`)
(4) グラフなどに plot して可視化 (`makegraph.sh`)
(5) 月が変わったら log file を 1 日単位で圧縮 (`archive.sh`)
(6) 一定の周期で (1)〜(5) を繰り返す (`table.cron`)

3 source file

改良後の logging system を構成するスクリプトを下記に示します．記述の都合上の理由で春号で記したスクリプト中のコメントや，それに関する解説は省略しています.[*10]

3.1 makedir.sh

```
1  #!/bin/bash
2  year=‘env LANG=C date +%Y‘
3  month=‘env LANG=C date +%m‘
4  day=‘env LANG=C date +%e | awk ’{print $1}’‘
5  timelabel=‘echo $year-$month-$day‘
6  mkdir $HOME/var/log/$timelabel
```

`date` コマンドをフォーマット付きで実行することで 46 行 → 6 行となりました．

3.2 recw.sh

```
1  #!/bin/bash
2  year=‘env LANG=C date +%Y‘
3  month=‘env LANG=C date +%m‘
4  day=‘env LANG=C date +%e | awk ’{print $1}’‘
5  prefix=$HOME/var/log/$year-$month-$day
6  prefix2=$HOME/local/bin
7  timelabel=‘env LANG=C date‘
8  hhmmss=‘echo $timelabel | awk ’{print $4}’‘
9  hh=‘echo $hhmmss | awk -F: ’{print $1}’‘
```

[*9] `ied{0...2}`→`ied{1...4}`となり，常時 4 つのホスト全てが起動しているようです．C 科 O 研の T 助教も御用達です．
[*10] 2007 年春号は MMA 公式 Web サイト (`http://delegate.uec.ac.jp:8081/club/mma/`) の `DOCUMENTS` からご覧ください．

```
mm=‘echo $hhmmss | awk -F: ’{print $2}’‘
ss=‘echo $hhmmss | awk -F: ’{print $3}’‘
hmm=‘$prefix2/comprate 60 $mm‘
hss=‘$prefix2/comprate 3600 $ss‘
time=‘echo "scale=6; $hh + $hmm + $hss" | bc‘
cd $prefix
Logging(){
    sort tmp-tenpu > tenpu
    grep pts tenpu > pts
    hitonokazu=‘wc -l pts | awk ’{print $1}’‘
    grep dtlocal tenpu > dt
    dtninzu=‘wc -l dt | awk ’{print $1}’‘
    awk ’{print $1}’ pts > tmp-pts
    awk ’{print $1}’ dt > tmp-dt
    uniq tmp-pts uni-tmp-pts
    uniq tmp-dt uni-tmp-dt
    diff uni-tmp-pts uni-tmp-dt | grep "<" | awk ’{print $2}’ > uni-tmp-remote
    j=0
    for i in "pts" "dt" "remote"
        do
        max=‘wc -l uni-tmp-$i | awk ’{print $1}’‘
        if [ $1 = 1 ]; then
            if [ $max -ne 0 ]; then
                logins=‘head -n $max uni-tmp-$i‘
                hoge=($logins)
                g=$i
                if [ $i = "remote" ]; then
                    i="pts"
                fi
                for ((cnt=0;cnt<max;cnt++))
                    do
                    grep "${hoge[$cnt]}" $i >> "${g}_${timelabel} ${max}"
                done
            else
                touch "${g}_${timelabel} ${max}"
            fi
        fi
        Dat[$j]=‘echo $max‘
        j=‘expr $j + 1‘
    done
}
for k in 1 2 3 4
do
    rsh ied$k env LANG=C w -hl >> tmp-tenpu
done
Logging 1
echo "$time ${Dat[@]} $hh:$mm:$ss" >> $prefix/time-login-all.dat
for k in 1 2 3 4
do
    rsh ied$k env LANG=C w -hl > tmp-tenpu
    Logging 0
    IedVec[$k]=‘echo ${Dat[@]}‘
    echo "$time ${IedVec[$k]} $hh:$mm:$ss" >> $prefix/time-login-ied$k.dat
done
rm pts dt
for j in 1 2 3 4
    do
    Pts=‘echo ${IedVec[$j]} | awk ’{print $1}’‘
    echo $Pts >> pts
    Dts=‘echo ${IedVec[$j]} | awk ’{print $2}’‘
    echo $Dts >> dt
    Rem=‘echo ${IedVec[$j]} | awk ’{print $3}’‘
    echo $Rem >> remote
done
for j in pts dt remote
    do
    Tmp=‘head -n 4 $j‘
    TmpVec=($Tmp)
    max=${TmpVec[0]}
    for ((cnt=1;cnt<=3;cnt++))
        do
        if [ $max -lt ${TmpVec[$cnt]} ]; then
            max=${TmpVec[$cnt]}
        fi
    done
    echo "$time ${TmpVec[@]} $hh:$mm:$ss $max" >> $prefix/time-login-$j.dat
done
rm tenpu tmp-tenpu dt pts remote tmp-pts tmp-dt uni-tmp-remote
rm uni-tmp-pts uni-tmp-dt
```

関数機能で構造化したり[*11]for 文を使うことで機能が増えた上で，136 行 → 88 行となりました．recw.sh では

- IED の全てのホストでの総ログイン数を pts，dtlocal，remote の形式で取得 (56 行目)
- IED の各ホストに関して，ログイン数を pts，dtlocal，remote の形式で取得 (62 行目)
- pts，dtlocal，remote に関して，ログイン数を各ホストごとに取得 (85 行目)

という機能が実装されています．シェルスクリプトでは 2 次元以上の多次元配列の機能がないので 61 行目で IedVec という擬似的 2 次元配列を構成しました．ログイン数を各ホストごとに取得するにあたって，別次元で配列を走査する必要があるためです．

また，85 行目で time-login-{pts,dt,reote}.dat を出力する際には後述の makegraph.sh にてグラフを生成する際に便宜を図る為，各ホスト中の最大のログイン数$max を付記しています．(65〜84 行目)

3.3 recloadaverage.sh

```
 1  #!/bin/bash
 2  year=‘env LANG=C date +%Y‘
 3  month=‘env LANG=C date +%m‘
 4  day=‘env LANG=C date +%e | awk ’{print $1}’‘
 5  prefix=$HOME/var/log/$year-$month-$day
 6  prefix2=$HOME/local/bin
 7  timelabel=‘env LANG=C date‘
 8  hhmmss=‘echo $timelabel | awk ’{print $4}’‘
 9  hh=‘echo $hhmmss | awk -F: ’{print $1}’‘
10  mm=‘echo $hhmmss | awk -F: ’{print $2}’‘
11  ss=‘echo $hhmmss | awk -F: ’{print $3}’‘
12  hmm=‘$prefix2/comprate 60 $mm‘
13  hss=‘$prefix2/comprate 3600 $ss‘
14  time=‘echo "scale=6; $hh + $hmm + $hss" | bc‘
15  cd $prefix
16  Logging(){
17      whead=‘rsh ied$1 env LANG=C w -u‘
18      vIed=($whead)
19      sup=‘echo "${#vIed[@]}"‘
20      i=‘expr $sup - 1‘
21  }
22  for j in 1 2 3 4
23      do
24      Logging $j
25      cnt=‘expr $j - 1‘
26      Ied[$cnt]=‘echo ${vIed[$i]}| cut -c 1-4‘
27  done
28  echo "$time ${Ied[@]} $hh:$mm:$ss" >> $prefix/time-loadaverage.dat
```

w -u や uptime コマンドでは “現在時刻 システムの総起動時間 ログイン数 平均負荷率 (load average)” といった出力が得られるのですが，システムの総起動時間は出力されないことがあります．なので load average が位置するフィールドは時可変です．そのため賢く設定する必要があります．

recloadaverage.sh では各ホストごとに 5 分間平均負荷率を取得しています．w -u の出力を配列変数に格納してフィールドごとに分割し (17〜18 行目)，5 分間平均負荷率が常に後ろから 2 つ目のフィールドに位置するという性質に着目して，配列の成分数を得て (19 行目)，着目する添字を設定し (20 行目)，整った体裁で記憶します (26 行目)．全てのホストで処理が終わったら Gnuplot 用にプロットデータを出力します (28 行目)．

本気で loadaberage を取得するのであれば w コマンドの source code を読み，/usr/include/stdlib.h の getloadavg 関数等を用いて自力で実装する方法をオススメします．

3.4 makegraph.sh

```
 1  #!/bin/bash
 2  year=‘env LANG=C date +%Y‘
 3  month=‘env LANG=C date +%m‘
 4  day=‘env LANG=C date +%e | awk ’{print $1}’‘
 5  prefix=$HOME/var/log/$year-$month-$day
 6  prefix2=$HOME/logging
 7  cd $prefix
 8  ArgMax(){
 9      if [ $2 = 0 ]; then
10          awk ’{print $2}’ $prefix/time-login-$1.dat > tmp-pts
11      else
12          awk ’{print $7}’ $prefix/time-login-$1.dat > tmp-pts
13      fi
14      NumLine=‘wc -l tmp-pts | awk ’{print $1}’‘
15      NumLogins=‘head -n $NumLine tmp-pts‘
16      VecNumLog=($NumLogins)
```

[*11] 出力を内包していることから Pascal 的な立場からすると手続き (Procedure) と称するのが自然だと思います．

```
17      tmpmax=${VecNumLog[0]}
18      for ((cnt=1;cnt<NumLine;cnt++))
19          do
20          if [ $tmpmax -lt ${VecNumLog[$cnt]} ]; then
21              tmpmax=${VecNumLog[$cnt]}
22          fi
23      done
24      max=‘echo "scale=2; $tmpmax + 0.3" | bc‘
25  }
26  for i in all ied1 ied2 ied3 ied4
27      do
28      ArgMax $i 0
29      /usr/local/bin/gnuplot << EOT
30          set term post color eps
31          set out "$prefix/time-login-$i.eps"
32          set key top right
33          set key box lt 1 lw 2
34          set style line 1 lt 1 lw 2
35          set grid xtics ytics mxtics mytics
36          set xtics 0, 3, 24
37          set mxtics 3
38          set mytics 5
39          set xlabel "Time [hour]"
40          set ylabel "Logins"
41          plot [0:24] [-0.3:$max] "$prefix/time-login-$i.dat" using 1:2 with lines ti "pts",\
42          "$prefix/time-login-$i.dat" using 1:3 with lines ti "dtlocal",\
43          "$prefix/time-login-$i.dat" using 1:4 with lines ti "remote"
44          set term png
45          set out "$prefix2/png/login-state-$i.png"
46          replot
47          exit
48  EOT
49  done
50  for i in pts dt remote
51      do
52      ArgMax $i 1
53      /usr/local/bin/gnuplot << EOT
54          set term post color eps
55          set out "$prefix/time-login-$i.eps"
56          set key top right
57          set key box lt 1 lw 2
58          set style line 1 lt 1 lw 2
59          set grid xtics ytics mxtics mytics
60          set xtics 0, 3, 24
61          set mxtics 3
62          set mytics 5
63          set xlabel "Time [hour]"
64          set ylabel "Logins"
65          plot [0:24] [-0.3:$max] "$prefix/time-login-$i.dat" using 1:2 with lines ti "IED1",\
66          "$prefix/time-login-$i.dat" using 1:3 with lines ti "IED2",\
67          "$prefix/time-login-$i.dat" using 1:4 with lines ti "IED3",\
68          "$prefix/time-login-$i.dat" using 1:5 with lines ti "IED4"
69          set term png
70          set out "$prefix2/png/login-state-$i.png"
71          replot
72          exit
73  EOT
74  done
75  /usr/local/bin/gnuplot << EOT
76  set term post color eps
77  set key top right
78  set key box lt 1 lw 2
79  set style line 1 lt 1 lw 2
80  set grid xtics ytics mxtics mytics
81  set xtics 0, 3, 24
82  set mxtics 3
83  set mytics 5
84  set out "$prefix/time-loadaverage.eps"
85  set xlabel "Time [hour]"
86  set ylabel "Load Average"
87  plot [0:24] "$prefix/time-loadaverage.dat" using 1:2 with lines ti "IED1",\
88  "$prefix/time-loadaverage.dat" using 1:3 with lines ti "IED2",\
89  "$prefix/time-loadaverage.dat" using 1:4 with lines ti "IED3",\
90  "$prefix/time-loadaverage.dat" using 1:5 with lines ti "IED4"
91  set term png
92  set out "$prefix2/png/la.png"
93  replot
94  exit
95  EOT
96  rm tmp-pts
```

67 行 → 96 行になりました．makegraph.sh では

- IED の全てのホストでの総ログイン数 (pts，dtlocal，remote)
- IED の各ホストでのログイン数 (pts，dtlocal，remote)
- pts，dtlocal，remote シグナルでの，各ホストごとのログイン数
- IED の各ホストの load average

のデータを時系列に記録されたプロットファイルを読み込み，EPS 形式と PNG 形式でグラフを出力します．尚，このスクリプトを実行する前にホームディレクトリ以下に logging というディレクトリを作っておかないと正しく動作しません．

また，ログイン数推移のグラフを生成する際に ArgMax 関数 (8〜25 行目) で最大値を取得することで plot 時に明示的に上限と下限を宣言し，グラフの縁と重ならないようにしました．得られるグラフの一部を**図 3**-**図 6** に示します．サンプリングした日は IED で情報通信工学実験が行われるとある金曜日です．テーマは信号処理のようでした．

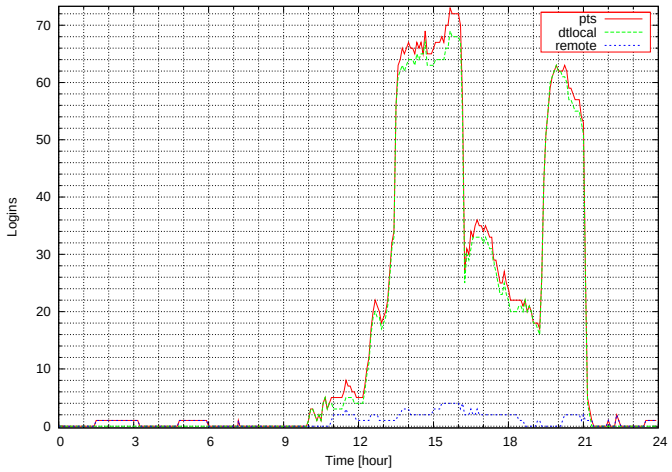


図 3 time-login-all.eps

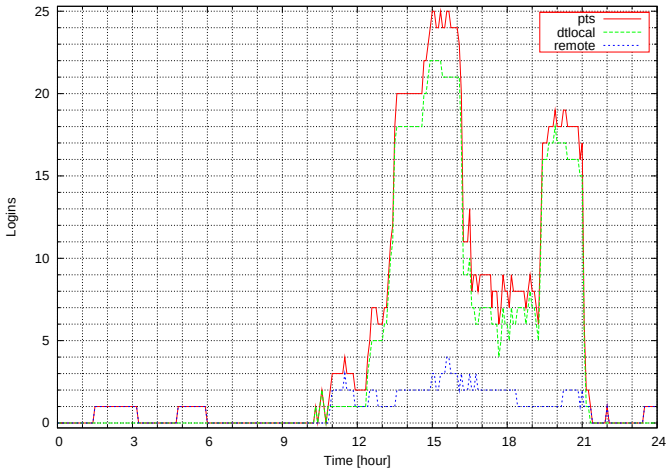


図 4 time-login-ied1.eps

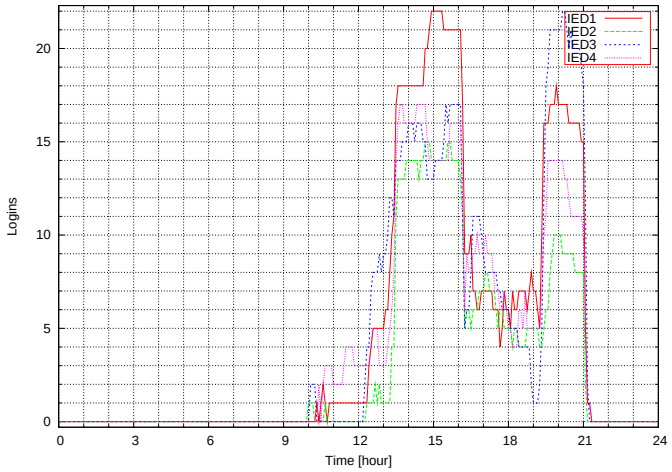


図 5 time-login-dt.eps

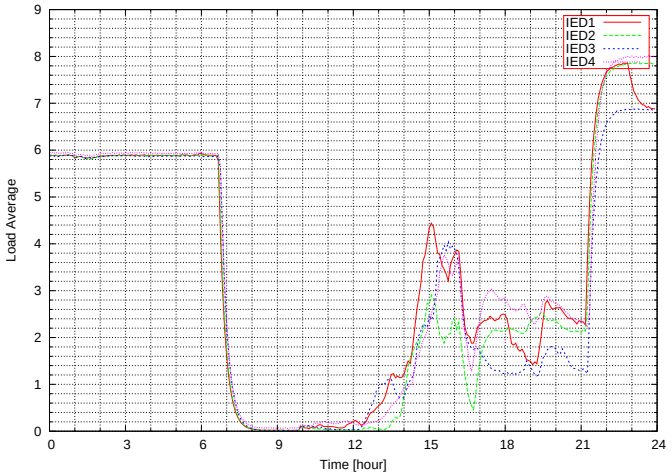


図 6 time-loadaverage.eps

各ホスト間で，ログイン数や loadaverage の推移に相関性があることが分かります．ローカルログイン数とリモートログイン数の推移の間には相関性が見れないので独立の確率分布に従っていると考えられます．

深夜や夕方以降にユーザが全てのホストで CPU 負荷をかけまくる並列演算プログラムを走らせている様です．ローカルログインの学生のプロセスにあまり影響を与えない時間なので，管理者は黙認しているのでしょう (このユーザ自体が管理者という可能性も否定できません)．

日中は信号処理実験というわけで Mathematica を上がって CPU 負荷がかかってていたようです．Mathkernel を暴走させていた学生も少なからずいたようです．こういう時にロクに制御できないのに gnome やらで xscreensaver[*12]などを上げっ放しにするのは個人的によろしくないと思っています．

[*12] twm と xlock で十分だと思いますが。。。

3.5 archive.sh

```
#!/bin/bash
year=‘env LANG=C date +%Y‘
month=‘env LANG=C date +%m‘
prefix=$HOME/var/log
if [ $month -eq 1 ]; then
    year=‘expr $year - 1‘;
    month=12
else
    month=‘expr $month - 1‘
fi
cd $prefix
ls -a | grep $year-$month | grep -v tar > list
max=‘wc -l list | awk ’{print $1}’‘
timestamp=‘head -n $Upper list‘
vtimestamp=($timestamp)
for ((cnt=0;cnt<max;cnt++))
    do
    /usr/sfw/bin/gtar cvjf ${vtimestamp[$cnt]}.tar.bz2 ${vtimestamp[$cnt]}
    rm -r $prefix/${vtimestamp[$cnt]}
done
rm list
```

makedir.sh 等と同様に日付変換部を省くことで 71 行 → 21 行になりました．機能は春号のものと代わりません．

3.6 table.cron

この table.cron は可読性が低いので機能のみ記述します．

(1) 毎日 0 時 0 分に makedir.sh を実行する．
(2) 5 分ごとに recw.sh を実行する．
(3) 5 分ごとに recloadaverage.sh を実行する．
(4) 新しい月に変わったら 1 時 0 分に archive.sh を実行する．
(5) 5 分ごとに sleep 5 した後に，makegraph.sh を実行する．
(6) グラフが生成されたら rsync で他サーバに転送する

こんな風な内容の crontab ファイルを書いて cron に登録することで logging system が完成です．尚，rsync 用にパスフレーズなしの公開鍵と秘密鍵の対を生成し, 転送先のサーバの ~/.ssh/authorized_keys に公開鍵を登録しておく必要があります．

4 おわりに

生成されたグラフの GUI 面での強化や他のスクリプト言語での実装，その他の需要のある機能の実装等が今後の課題です．

おまけ

```
/* comprate.c                                  */
/* usage: gcc -o comprate -O2 comprate.c  */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
        int a, b;
        double tmp;
        sscanf(argv[1],"%d",&a);
        sscanf(argv[2],"%d",&b);

        if(a!=0){
                tmp=(double)b/a;
                printf("%f\n",tmp);
        }
        else
                printf("Nice boat.\n");
}
```

編集後記

(著者) 百萬石編集人 moechar
moechar@mma.club.uec.ac.jp

本日は第 57 回 調布祭 MMA 展示室及び MMA ジャンク市にお越しいただき，誠にありがとうございます.

表紙について

表紙ページは私が作成しました．ついでに書きますと，調布祭パンフレットの折込チラシの基本的なデザインは私が担当しました.
左上からラスタ走査順に OpenBSD, FreeBSD, LaTeX, Gentoo Linux, Momonga Linux, GNU/Linux オペレーティングシステム, SUSE Linux, Vine Linux, Debian GNU/Linux のロゴマーク・マスコットキャラクターを配置しました。
これらロゴマークの下にある力強いメッセージは**メンズ ナックル**というファッション雑誌の 11 月号のインターネット上で話題になりつつある読者モデルの画像が引用元です．ただし，LaTeX の方では原文では，

エレガントに舞い，クレイジーに酔う

と記載されていたものに一部変更を加えました.

編集後の感想

ソースコードを掲載するにも投稿者によっていろいろ流派があるようで意外でした．学内には `verbatim` 環境を使ってソースを載せたレポートを書く学生が多いようです．もっと明解な組版結果がえられる解があるのにもったいないことです.
第 5 章では，私も筆をとらせてもらいしたが，編集が進むにつれて内容は稚拙であり文才も感じられないため MMA 的によろしくないなと感じました．そのため速やかに読み飛ばされることをオススメします.

次号に向けて

私としましては，投稿者には締切り以前の日時に余裕を持って脱稿してもらいなと願うのももちろん，さらに LaTeX 表現方法の熟達に励みたいと思っています．また普通の書籍のような奥付のページも巻末に設けてみたいと思っています.

MMA 公式 Web page

http://delegate.uec.ac.jp:8081/club/mma/