百萬石

百萬石 2005年秋号
MMA[1]

目 次

[1] みんなで マイコンで 遊ぼう会

第 I 部
ラプラス解析による EMI フィルタ回路の設計

(著者) J/0321012, 仲田将之

センサが出力するほとんどの情報は微弱なものだ. 今回はある種のセンサが出力する振幅が数 mV ほどの微小な情報を, 外乱ノイズに消されることなく, 測定したい. この外乱ノイズを取り除く回路のことを一般に EMI[2] フィルタとよぶ. 図 1 はその基本的な回路だ.

今回は風速計が出力する正弦波信号の周波数を正負の電位でコンパレートして測定する. この波の振幅は誘導起電力であるために周波数に比例して変化し, 風速 0.5m/s を意味する周波数 5Hz では振幅 80mV だったものが, 風速 50m/s の周波数 500Hz では振幅 8V にもなる. 高圧信号が入力されては測定機器が破壊されるので, どの周波数の信号が入力されても, EMI 回路から出力される振幅を電源電圧の 5V を上回らないように制限したい. また, 単電源の測定機器を使うので, 出力の基準電位を電源電圧の中間電位である 2.5V にするために入力側にカップリングコンデンサを入れたい. しかし, この影響で低周波の信号が減衰してしまうので, それを軽減するような容量のカップリングコンデンサを選びたい. また, 測定機器を過電流から保護するために, 数 kΩ ほどの電流制限抵抗を入れたい.

これらの条件を全て満して, しかもセンサの信号に対しては影響を与えないようなパラメータを持つ EMI フィルタを定めたい. 一般に EMI フィルタは差動信号も同相信号も減衰するが, センサから出力される情報は差動信号なので気になるのは差動信号の減衰をいかに少なくできるかということだ. そこで, 図 2 のように差動信号のフィルタに簡略化した回路について考えることにする.

電流 i が抵抗 R, キャパシタ C を流れたとき,

$$e_R = Ri$$
$$e_C = \frac{1}{C}\int idt$$

なる電圧 e_R, e_C がそれぞれに掛かる. したがって, 図 2 の回路に対して, 電圧 $e_i(t)$ を入力信号, $e_o(t)$ を出力信号, 電流 $i(t)$ を回路に流れる電流とすると,

$$e_i(t) = Ri(t) + \frac{1}{C}\int i(t)dt + e_o(t)$$
$$e_o(t) = \frac{1}{C_o}\int i(t)dt$$

がなりたっている. 電流 i を置き換えたいので,

$$\frac{de_o}{dt} = \frac{i}{c_o}$$

と微分すれば,

$$e_i = C_oR\frac{de_o}{dt} + \frac{C_o}{C}e_o + e_o$$

となる.

$e_o(t=0) = 0$ として, それをラプラス変換すると

$$E_i = C_oRsE_o + \frac{C_o}{C}E_o + E_o$$
$$= (C_oRs + \frac{C_o}{C} + 1)E_o.$$

伝達関数とよばれる $G(s)$ を

$$G(s) = \frac{E_o}{E_i} = \frac{1}{CRs + \frac{C_o}{C} + 1}$$

で定義しよう. 伝達関数 $G(s)$ を使えば, L^{-1} を逆ラプラス変換とするとき, 出力信号 $e_o(t)$ を

$$e_o(t) = L^{-1}\left(E_o(s)\right) = L^{-1}\left(G(s)E_i(s)\right)$$

として, 入力信号のラプラス変換 E_i から求めることができる. 入力信号 $e_i(t)$ が α を係数とする正弦波:

$$e_i(t) = \frac{1}{\alpha}\sin(\alpha t)$$

であるとすると, そのラプラス変換 L は

$$E_i(s) = L\left(e_i(t)\right) = \frac{1}{s^2 + \alpha^2}$$

と分かっているから, したがって,

$$E_o(s) = G(s)E_i(s) = \frac{1}{C_oRs + \frac{C_o}{C} + 1}\frac{1}{s^2 + \alpha^2}$$

を部分分数に分解すれば, $C_a = \frac{C_o}{C} + 1$ とすると,

$$= \frac{1}{(CR\alpha)^2 + C_a^2}\left[\frac{(C_oR)^2}{C_oRs + C_a} + \frac{-C_oRs + C_a}{S^2 + \alpha^2}\right].$$

したがって,

$$e_o(t) = L^{-1}\left(G(s)E_i(s)\right)$$
$$= \frac{1}{(CR\alpha)^2 + C_a^2}\left[C_oR\exp\left(-\frac{C_a}{C_oR}t\right)\right.$$
$$\left. -C_oR\cos(\alpha t) + \frac{C_a}{\alpha}\sin(\alpha t)\right].$$

[2] Electric Magnetic Interference: 電磁干渉

f を周波数として $\alpha = 2\pi f$ とすれば

$$e_i(t) = \frac{1}{2\pi f}\sin(2\pi f t)$$

であるが, 目的は振幅が周波数に比例する信号を入力することだった. 今回は 62.5Hz のときに振幅を 1 にしたいので $k = 62.5$ として, 振幅を周波数 f に比例する値 f/k とした. つまり

$$e_i'(t) = \frac{\alpha f}{k} e_i(t)$$

がその入力信号となるということだ. e_i と e_o は線形な関係にあるので

$$e_o'(t) = \frac{\alpha f}{k} e_o(t)$$

がその出力信号となる. ここで求めたかったことは, 出力信号の振幅であったから, $e_o(t)$ の第 1 項にある $\exp(-\beta t)$ なる減少関数を無視し, またその第 2, 3 項については, $\gamma = \sqrt{(C_a/\alpha)^2 + (C_o R)^2}$ とし, $sin\delta = -C_o R/\gamma, \cos\delta = C_a/\alpha\gamma$ なる δ をおくと,

$$\begin{aligned}&\frac{C_a}{\alpha}\sin(\alpha t) - C_o R\cos(\alpha t)\\ =&\gamma\left(\frac{C_a/\alpha}{\gamma}\sin(\alpha t) - \frac{C_o R}{\gamma}\cos(\alpha t)\right) = \gamma\sin(\alpha t + \delta)\end{aligned}$$

である. したがって, 周波数 f において, 入力信号 $e_i'(t)$ に対する出力信号 $e_o'(t)$ の振幅は:

$$\begin{aligned}\frac{\alpha f}{k}\frac{\gamma}{(C_o R\alpha)^2 + C_a^2} &= \frac{\alpha f}{k}\frac{1}{\alpha\sqrt{(C_o R\alpha)^2 + C_a^2}}\\ &= \frac{f}{k}\frac{1}{\sqrt{(C_o R\alpha)^2 + C_a^2}}.\end{aligned}$$

実際に図 2 の回路に定数 C, R, C_0 を入れてみよう. $k = 62.5$ であった. 保護抵抗 $R = 20k$, 振幅制限用のコンデンサ $C_o = 0.1\mu$, 直流カットコンデンサ $C = 0.5\mu$ としたときの周波数特性は図 3 となった. この図 1 から直流カットコンデンサの容量が小さすぎて, 低周波の信号が減衰していることが読み取れる. したがって, 直流カットコンデンサ $C = 5\mu$ として計算しなおしたものが, 図 4 である. 低周波の減衰が抑えられていることが分かる.

参考文献として, Analog Device 社のアプリケーションノート AN-761 と, JSME テキストシリーズの制御工学を使った.

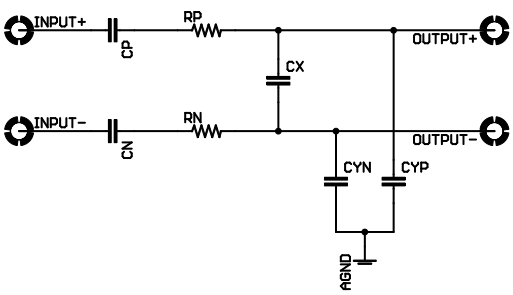

図 1: EMI フィルタ回路

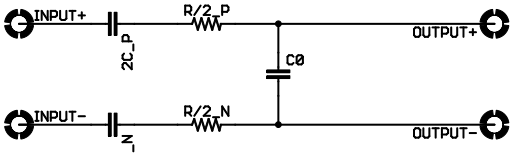

図 2: 差動信号に対する EMI フィルタ回路

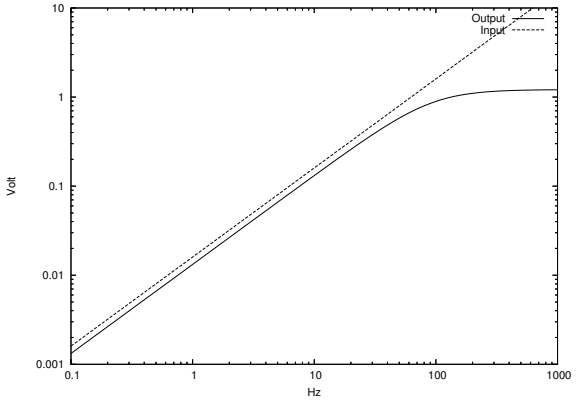

図 3: カップリングコンデンサの容量が 1uF のときの周波数特性

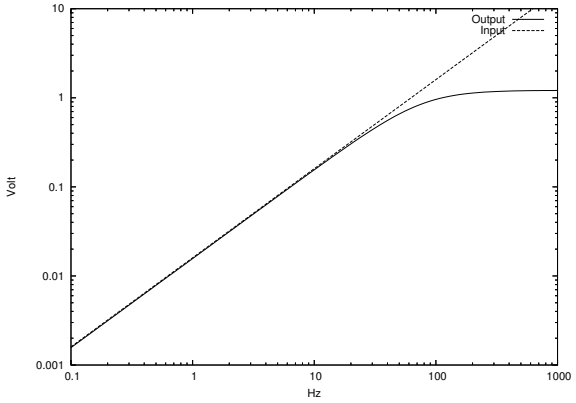

図 4: カップリングコンデンサの容量が 10uF のときの周波数特性

第 II 部
オペレーティングシステム cltn の制作 (その 1)

(著者) oku

今年は色々と間に合わなかったので、「その 1」って事にしています。来年は cltn の名前ができてから 10 年の節目なのでどうにかしたい。。

1 cltn とは何か？

cltn[3]は、oku が個人的に作っている OS で、要は自分が満足できるだけのコンピューティング環境を作れないだろうかという探求です。

1.1 なぜ OS か？

今の UI が気に食わないだけならば、例えば GUI だけ作ってカーネルやファイルシステム等は既存の OS の実装を用いるという方法も有りそうですが、既存の UI が気に食わないからこそ OS から実装しようという動機になります。

cltn では、「ユーザインターフェースはプログラムではなくデータに与えられるべき」というアイデアを基本に持っています。これは Smalltalk やその他のオブジェクト指向環境の Model-View-Controller 分割にインスパイヤ[4]されたものです。

既存のオペレーティングシステムの殆どは、プログラムをエンドユーザに作成/変更させることを意図しているわけではなく、UI を作り直すことすら簡単では有りません。これは、プログラムがどのようなデータを取り扱っているのかを把握していないことに起因する問題で、これを解決するためにはプログラミング言語、ライブラリを含めたプログラミング環境全体を OS に合わせて作る必要があります。

1.2 既存の環境の考察

もちろん、既存の環境の良い部分、悪い部分を知ることは有益な事です。

1.2.1 Macintosh (classic MacOS)

古き良き Macintosh は UI をプログラムコードから分離したリソースの形で持ち、ResEdit などのアプリケーションを用いることでアプリケーションの UI を作り変えることができました。この特徴は、Windows にもあてはまりますが、プログラムからの独立性という意味では Macintosh のそれは特徴の一つに数えられる程[5]です。

また、ファイルタイプの概念をファイルシステム自体が持っていて、ファイルタイプとアプリケーションの関連をシステムが管理できました。これは POSIX がデータと実行ファイルを区別したり、Windows がファイル拡張子で行うものとは別次元のものです。Macintosh におけるファイルタイプはユーザから隠蔽されていた (ユーザはファイルタイプをアイコンの形状で意識する) ので、アプリケーションベンダは安心してアプリケーションを提供することができました。

平たく言えば、

- プログラムコードと UI は分離するべきである

[3]「くるとん」と発音します
[4]このアイデアを他の環境に適用することを妨げるものではありません
[5]ResEdit 一本でシステム全体を日本語化できる程という話もある

1.2.2 AppleEvent、OLE

Macintosh System7 で導入された AppleEvent は、アプリケーションの動作に名前を付け、アプリケーションの UI 以外によるコントロールを可能にしたものです。

Windows には OLE という仕組みがあり、アプリケーションが共通のプロトコルでオブジェクトをやりとりすることができます。

普通のユーザもこのような仕組みを用いて無意識のうちに異るアプリケーション同士を連携させて利用しています。つまり、オブジェクトをコピーして、他のアプリケーションに貼り付けることで、アプリケーション同士を連携するのは良く見られる光景です。

- プログラムの動作をシステムに認識させ、個別 UI に頼るべきではない

これらの仕組みは「アプリケーションを組み合わせて環境として使える」という点で共通した思想を持っています。が、このような仕組みは Smalltalk のようなプログラミング環境とオペレーティング環境の融合したものでは不要なものです。

1.2.3 Smalltalk、GNU EMACS、Plan9

Smalltalk は言わずと知れた GUI の祖の一つですが、プログラミング環境とオペレーティング環境が正常に融合することに成功した成功例でもあります。

システムの要素を統一したインターフェースでユーザに制御させるために、Smalltalk はシステムの要素をオブジェクトの概念にとじこめ、その制御手段をオブジェクトに対するメッセージの送信という形で統一する[6]ことで一つのルールがより広範に適用できるようになっています。

プログラミング環境と生活環境が同一のシステムとなっているものとしては、他に GNU EMACS があります。これはシステムの殆どが EMACS LISP とよばれるプログラミング言語で構成されていて、ユーザは LISP を通してシステムの殆どの要素にアクセスすることができます。もちろん、いわゆるマクロ機能を供えることでプログラムから制御できるエディタは数多くありますが、GNU EMACS は OS であるという意見もあるほどその自由度は高く、かな漢字変換やメーラ、Web ブラウザに至るまで EMACS の中に実装されています。

Plan9 は、システムの要素をファイルの概念に押しこめています。これはシステムの拡張性をいちじるしく向上させます。

CPU のファイルを共有すれば分散コンピューティングが可能に、Window の表示内容もファイルに見えるので、それをコピーすればスクリーンショットがとれたりなど。

また、ファイルの概念も普通の OS に比べて拡張されていて、「昨日の時点のファイル hoge を開く」のようなオペレーションも可能になっています。

- システムの要素を統一された概念でとりあつかう

1.2.4 超漢字 (BTRON)

BTRON はファイルシステムにおいてハイパーリンクとファイルタイプの定義を行う機能をもち、リンクの編集はシステム標準の UI でおこなうことができる。

ファイルに関連付けられたファイルタイプにより、ユーザはアプリケーションの存在を意識することは無く、データをともなわずに起動されるアプリケーションは小物という形でシェルのサブメニューに収まることになります。

ただし、BTRON のファイルは上で書いたようなシステム概念としてのオブジェクトからは幾分と縮退した形で、デバイスなどの書類でないものはユーザからはほぼ完全に隠蔽されています。ファイルは TAD(TRON Application Databus) として共通化され、アプリケーションは TAD エディタという位置付けになります。

[6]Smalltalk の VM では整数演算などいくつかのプリミティブはバイトコードにより直接実行される。これはいうまでもなくパフォーマンスのため

1.3 理想のコンピューティング環境の条件とは

単純に「あったらいいな」を列挙すれば、

- ユーザが自由にソフトウェアを分解できる
- Write Once Run Anyware
- 過去にも戻れるリアルタイム性
- (スクリプト言語における) ワンライナーの自由度

これらを実現する魔法の鍵は「オペレーティング環境とプログラミング環境を統一する」ことと言えます。

で、今の環境がオペレーティング環境とプログラミング環境の乖離したものかというと「プログラマにとっては」統一されていると言えます。その証拠に UNIX オペレーティングシステムでは大抵"man fopen"のようにすることで API のマニュアルを参照することができます。

しかし、ユーザにとっては乖離したものといえて、それはいいかたを変えればプログラムが完全にブラックボックスになっている[7]状況であるということです。

その状態を「コンピュータを使っている」と果して言えるかどうかと。

コンピュータを我々の手に取りもどすためには、全てのユーザがプログラマでもある必要があります。当然、通常の人間は UNIX プログラマにはなりませんから、より理解しやすく手軽なプログラミング環境でオペレーティング環境を統一する必要があります。

1.4 Java やスクリプト言語のようなアプローチと OS

ユーザをプログラマにするための手軽な手段がスクリプト言語などのアプローチです。また、Java や Smalltalk も既存の OS の上に統一されたレイヤを作って目的をよく達成しています。

これらのアプローチの問題点は効率に集約されます。つまり、OS の動作を既存の OS 上で実装する事は可能ではあるが、エミュレーションのためのオーバーヘッドが有りそうなことは容易に理解できます。

ソフトウェアの (実行) 効率を突き詰めるとそれは OS になります。

2 cltn におけるプログラミング

2.1 プログラム要素の分割

記述や実行の効率を考えれば、システムの持っている問題の分析手法、つまり、プログラミングパラダイムは多ければ多いほど良い[8]ことになります。cltn では、「状態遷移」「アルゴリズム」「論理ゲート」「関数」の 4 つをとりあえず想定して UI を作成することにしています。

「とりあえず」というのは 2 つの意味があって、これから増える可能性と、「アルゴリズム」以外は最適化などの考察を一切してないという意味です。

システムがプログラミングパラダイムを多く備えることは、実行効率はもちろん、記述効率にも寄与します。例えば、状態遷移を用いて平行プログラミングを行えば、事前にデッドロックなどの設計上の問題点をチェックすることが可能になります。

[7] マクロで操作可能であったり、AppleEvent のように構造が露出しているところにのみ光が当っている
[8] マルチパラダイムなプログラミング&実行環境としては竹内センセイらの TAO が有名か

2.2 カラー

cltn は可能な限りプログラムに UI を関連づけないので、プログラムの記述はコンピュータに対して設計を伝達するだけでなく、記述されたソフトウェアを利用するユーザにもプログラムの情報を伝達することが必要になります。

そこで、cltn におけるプログラミングではカラーと呼ばれる属性を自由にオブジェクトに対して付与できるようにし、コンピュータの「認識」にも名前を付けます。

たとえば、ロックが解除されていないとドアが開かない というのを、状態遷移と論理ゲートを用いて記述すると、

```
-CanOpenDoor = Unlocked(LOCK)

state Door : DoorOpen DoorClose
+OpenDoor = (CanOpenDoor) DoorClose to DoorClose

color locked(LOCK) = descript:ja (ドアがロックされている)
```

LOCK の状態 locked にカラーを割りあて、「ロックされている」という説明文を付けています。もしも CanOpenDoor でないために OpenDoor できなければ、システムはユーザに対して理由を示すことができます。

いままで、プログラマはソースコードに対するコメントの形で、コンピュータの認識と人間の認識を関連づけていましたが、cltn ではカラーに対してそれを行います。

また、ユーザはオブジェクトに付与されているカラーを操作することでプログラムに対する指示とすることができ、プログラムがオブジェクトに付与したカラーを観察することで、プログラムの動作を認識することができます。

このように、カラーは人間の認識とコンピュータの認識をつなぐ役割を果すことになります。

2.3 (狭義の) プログラミング環境

狭義のプログラミング環境とは、ユーザがプログラミングするための UI を指すものとします。

全てのユーザの行動、つまり、プログラムへのデータ入力を含めて、全てをプログラミングの一環と捉えてシステムを構築する必要が有ります。別の言い方をすれば、ユーザもシステム中のオブジェクトの一つに過ぎないということです。

各々のプログラミングパラダイムに対して、必要十分な機能を持ったエディタを用意しなければ現実的な手間ではプログラミングすることは出来ない、逆にいえば、ソースコードをテキスト文書として書かせることはしません。

むしろ、ソースコードがテキスト文書である必然性は一切ありません。

3 オペレーティングシステムとしての cltn

オペレーティングシステムとしての cltn はプログラムをメッセージの集合に変換し、適切なデバイスに適切なタイミングで投げるのが仕事になります。

3.1 メッセージとスケジューリング

プログラムはメッセージに変換されて実行されます。また、コンピュータの外部環境からの入力もすべて同じメッセージとして扱われます。

実行環境から見ると、プログラムやデバイスはメッセージを受信してメッセージを送信するものということになります。

メッセージをどのようなタイミングで出力するかなどを設計することをスケジューリングといいます。これは普通のシステムがコンパイラとカーネルのスケジューラで行うものを、cltn ではスケジューリングとして一緒くたに扱います。

3.2 静的スケジューリングと構造の隠蔽

静的スケジューリングとは、通常のシステムではコンパイラで行う、プログラムをメッセージ列に変換する動作を指します。静的スケジューリングは全てを動的なスケジューリングで行うよりも構造の隠蔽が可能であるため効率が高くなります。

cltn では静的スケジューリングのために、ハードウェア記述を用います。これは gcc の RTL のような CPU 記述で、オペコードごとに、CPU の実際に行う動作を記述するものです。ハードウェア記述はオペコードと付与されるカラーを対応付けたもので、これをもとにシステムはオブジェクトが目的のカラーを持てるように、CPU に対するメッセージを構築していきます。

多くのシステムではコンパイラが処理するのは CPU や周辺 DSP くらいのものですが、cltn の静的スケジューリングは USB デバイスのようなものに対するメッセージも生成します。つまり、デバイスドライバの類もハードウェアに対するメッセージと付与されるカラーの対として記述されることになります。

静的スケジューリングによって構造の隠蔽が行われます。つまり、すべての状態遷移を直接に観察できるわけではなくなります。構造の隠蔽が行なわれる基準は「UI が要求されるかどうか」ですが[9]、まだ良い UI 要求の手段を構築するには至っていません。

デバイス表現の一つの方法に、等価なプログラム表現とメッセージを対応させるものもあります。等価なプログラム表現をもっていると、静的スケジューリングの際に、命令スケジューリングのような高度な最適化を行うことも考えられます。

また、メモリの使用量を現実的なものに収めるために、同じ動作で共通して利用されるメッセージをライブラリのような形で適当にまとめます。現状では、纏める単位を明示する必要が有ります。

3.3 動的スケジューリング

動的スケジューリングは普通の OS ではカーネルのスケジューラが行うもので、装置から送られてきたメッセージを何処に配送するべきかを実行時にリアルタイムで判断するものです。

現状では、マルチプロセッシングにたいしての考察は無いので、単純に優先順位の付いたメッセージ配送機構ということになります。

また、後述するストリームスケジューリングのために、CPU イベントスケジューラと I/O スケジューラは独立しています。

3.4 リアルタイムストリームスケジューラによるメッセージ帯域保証

メッセージの帯域を保証すると一口にいっても、それなりに多くの要素が関わってきます。

帯域保証のニーズの有るメッセージには、

- ネットワークストリームなど非同期で帯域の一定しないもの
- HID の入力から画面更新までのような到達時間を保証したいもの
- HDD レコーディングのような物理デバイス特性によって影響を受けるもの

[9]普通のプログラミング環境だとデバッグ情報を捨てるかどうかと同じところ。つまり cltn における UI の意義にはデバッグインターフェースとして要求されるものも含む

など、様々なファクタがあり、完全に自動で最適なスケジューリングを行うのは難しいです。

しかし、ストリーム毎にスケジューリング優先度を設定できると、ユーザにより具体的な優先度設定を行わせることが可能になるので、既存のシステムのようにプロセス単位ではなく、たとえば耳に聞こえる音は途切れないで欲しい のような結果を軸にしたニーズを満すことが可能になります。

基本的に静的スケジューリングは Just In Time に行われますが、掛る時間を事前に予測することが非常に困難なため、必要なメッセージ群は予め生成されます。

4 cltn の UI 要素

4.1 アプリケーションレスデザインとストリームパッチベイ

超漢字を手本にした、アプリケーションレスの UI デザインを目標にしています。すなわち、アプリケーションの起動を意識することなく、データを直接に表示や編集の可能なものを。

Winamp のような編集アプリケーションで無いもののほとんどはデータベースのクエリ画面のようなものに分類できるので、「データのビジュアライザ」という分類を設けて対応しようとおもいます。

ストリームパッチベイは UNIX におけるパイプやリダイレクトのアイデアを拡張したもので、複雑な動作は細かい動作を組み合せて行おうとするものです。また、ストリームは任意の単位まで分解可能[10]なので、動作の調整をそこから行えます。ただし、リアルタイムストリームは静的スケジューリングを後から行うことはできないので分解も不可能ということになります。

ストリームは既存の OS においてプロセスに当る概念と言うこともできます。

5 まとめと次回予告

今迄の文章に書いてあるのは「過去に実装されていたことがある」というもので、例えば今のコードベースでは一度スケジュールされたストリームを再度静的スケジュールすることは直接にはできなかったりします。

現状ではコードベースをそれなりのペースで破棄して新しい機能を盛り込んでいる状況なので、その状況を改善するのが急務です。いきあたりばったりではなくて、ちゃんとした実装計画を持って。

この原稿に関しては、今回は「絵の出る VM(= GUI)」の実装が間に合わなかったので絵的に寂しい原稿になってしまいましたが、次は絵入りで。。

というわけで次回は UI デザインとかその辺の話を予定[11]。

[10] デバッガは CPU 直前の命令ストリームを T 分岐して実装する

[11] しかし予定は未定

第 III 部
日本人のメンタリティとWebの変革

(著者) 0410077 Tajima Itsuro(niryuu)
niryuu@series60.sakura.ne.jp

概要
それにしても長くなった。製本する手間が増えるではないか。しかし縮小して印刷すると稗圃に最低評価にされる。
日本人とは何か。なぜ個性をなくし「みんなと同じ」に走るのか。変革したインターネット。新しいパラダイムの中で日本人は変わるのか、それとも変わらないのかを独自の視点で斬る戦慄の未完作!

6　日本人の行動原理

6.1　常についてまわる「空気」

まず、この原稿での主題となる「日本人」について考えよう。私のような人間ではなく「普通」の日本人である。個性が尊重されているとされる現代社会でも「普通の人」というのは厳然と存在する。満員電車でよく見る背広のサラリーマン、彼らは自身も認める「普通の人」である。その辺で遊んでいるティーンエイジャーを見てみる。彼らは個性を自由に伸ばしているように見える。しかし、その会話を聴いてみると「みんな」「俺ら」と、いつも遊んでいる集団の中での同一性を重視しており、完全な個性はないように見える。彼らもまた「普通」である。このように世代を越えて存在する「普通の人」は、ほぼ完全に価値観、行動を共有している。その価値観を持たない人間を「普通ではない」という。場合によっては私のように「やばい」「怖い」と言われることもある。

普通の人は、とにかく周囲に合わせることを望み、それを実践している。それができない場合、集団から排除される危険性がある。その結果として見かけ上は価値観を共有しているように見える。普通の人の会話は、常に差し障りのない会話である。個人で違う部分については普通の人の会話に現れないためである。そのため、普通の人は決して授業で発言するようなことはせず、普通の人が壇上で挨拶などをするときは非常に中身の無いものになってしまう。

では、人はなぜ「普通」になるのだろうか。それを理解する鍵を故山本七平が独自の視点で研究している。著書「空気の研究」の中で氏は日本人に「空気」という力が働いているとしている。この抽象的な言葉はしばしば会話で出てくる。先に「周囲に合わせる」と書いたが、実際は「周囲」は特定の集団の意思決定などではなく、この「空気」である。

「空気」は会話による論理的帰結ではなく、会話を交わすこと自体により生成される。会話の中である意見が頻繁に現れると、それの妥当性にかかわらずその意見は「空気」となり、日本人を支配する。その例として氏は太平洋戦争を挙げている。政府や軍部を含みほぼすべての国民が敗戦を確信していたにもかかわらず、既に戦争継続の「空気」が醸成されていたため相当の悪条件で戦争を終わらせることになった。そのころ講和をできなかった理由を論理的に説明することはできない。

よって、特定の方法によって人工的に「空気」を醸成することも可能である。その方法を大手メディアは熟知しており、氏は別の著書でメディアを「日本教の教祖」と表現している。教祖の名の通り、自然発生が非常に難しい社会全体での「空気」はメディアによる物が多い。その例として少し前の韓流ブームを挙げられる。これはメディア（というよりメディアに強力な影響力を持つ広告代理店）が作った人工的ブームである。しかしインターネットにより疑問視され、当初計画していたとおりの規模にはならなかった。日本テレビなどはその時期に得た大量の韓国ドラマ放映権を事実上の不良債権として抱えている。

「空気」が日本人特有であることの理由として氏は宗教の違いを挙げている。一神教では全ては神に対して相対化されるため、相対的概念として把握できる。しかし日本の場合は物質に神が宿ると言う考えが潜在的に存在し、物質や事象に対して臨在観的把握（何かが存在する感覚、一種の感情移入）をし、絶対化してしまう。それにより他との比較ができなくなり、対立意見である論理的判断を妨げるというのである。一神教ではそれを避け

るために偶像を禁止する場合が多い。もっとも最近は海外でも一神教の影響力が弱まることも多く（むしろ支配層は回帰している）、日本特有でもなくなって来ている。

6.2 和の精神と村社会

さて、このような「空気」が浸透している日本ではどのような社会が形成されているのだろうか。日本人は、特定の集団で集団内のメンバーに対し、集団全体の空気を読めているかどうかを相手に踏み絵を差し出すことで判定する。踏み絵は「みんなと同じところに遊びに行くか」から始まり「戦争を継続するべきか」まで多岐にわたる。踏み絵の判断のため日本人は人を評価するときは結果ではなく如何に行動したかが基準になる。

集団の中で互いに踏み絵を出し合うことで結果的に集団内での統一された価値観、判断基準が生まれる。これはその場その場の空気由来のものにとどまらず、古くからの伝統や「常識」といったものまでの集合体である。世代などの大きな集団での踏み絵による価値観を持った人が「普通」の人であり、このような集団を村社会と呼ぶ。村社会は日本的な企業にもよく見られる。全社的価値観の認知から「ノミュニケーション」「窓際社員」まで村社会の特徴をそろえた企業は多い。

村社会の踏み絵による相互監視の結果表面上は考え方が統一されたため、人の気持ちを察することができるという幻想が生まれた。その結果、気持ちを察して行動することがよいこととされた。それが「思いやり」やいわゆる「やさしさ」である。これは実際には相手のためになるものではなく、むしろそれに対する妥当なレスポンスを期待し（好意的でない結果だと空気を読めていない）、しかも相互監視により常に気持ちを察して行動することが常識とされ、強いられる。気持ちを察しているかどうか自体が踏み絵になるのだ。

そのような村社会の中で「普通」でない異質な存在が入ってきたらどうなるか。踏み絵を踏めず、思いやりも効果がない、そんな人間を村社会は「村八分」にし、排除する。そうなるともはやその集団にはいられない。地域社会や義務教育など半ば強制された集団にいる場合は最悪である。無視されるならまだいい。「いじめ」が発生し、しかもいじめるかどうかが踏み絵になることで集団内の人が次々といじめに参加する。最悪の場合は自身が耐えられなくなり、引越しや不登校はもとより凶悪犯罪や自殺に走る場合さえある。もっとも今まで苦労しながら集団内の立場を維持してきた人間が突然閾値を越えてこのようになる場合もある。少年犯罪の際の担任や生徒のインタビューなどに「普通の子」というフレーズを良く聞く。河原美代子もあの地に来る前は普通だったという。

6.3 日本でしか根付かない萌え

氏は物質に対してのみ考察しているが、「空気」はもっと多くのものを説明できると思われる。その例として、近年「電波男」を出版し「萌え」の評論によって一躍時の人となった本田透氏の考え方が挙げられる。それを以上の議論で考察すると、日本における恋愛と萌えは日本特有のことであることが分かった。

氏は、そもそも恋愛は比較的最近登場した概念であるとしている。以前は西洋においては神により自我と男女の関係性が保証されていたが、ルネサンスから産業革命にわたる「神の死」によりそれができなくなった。そこで男女が相互に相手を絶対化することで、つまり恋愛によって自我を保証することになったという。つまり、この論法によれば恋愛は相互の臨在観的把握にほかならず、論理的に判断することができないものであることがわかる。

この恋愛が日本に渡り、そのうちに恋愛結婚が見合いに取って代わった。そして 80 年代、バブル期に女性の社会進出が本格化したこともあり、恋愛が一気に商品化した。前述したように日本人の価値観は一定であるため、その評価基準で恋愛の可否を判断することになる。結果として恋愛という観点では自由な趣味は排除され、特定の価値観を多くアウトプットする人間のみが生き残ることになる。その結果を氏は「恋愛資本主義」と呼んでいる。そしてそこからあぶれた一部が萌えに走っているということである。

一方で「萌える」人々も相当日本人的である。そもそも萌えは（広義の）記号に対する愛情ということになる。愛情は前述の通り対象への臨在観的把握なので、萌えは「空気」の醸成過程と非常に似ていることになる。このためキャラクターは絶対化され、自分だけの存在となり心の支えとなる。結局萌えは日本特有のメンタリティに基づくと考えられる。同人サークルなどの萌えるオタクのコミュニティも日本的な村社会を形成している。彼ら

は閉鎖的で仲間意識が強く、サークルのために個々がベストを尽くすことを要求される。そこには確固たる統一された価値観がある。一方オタクという大きな集団で見ると踏み絵はたとえば深夜アニメの名前をいくつか知っているだけでパスできるので私は非常に楽だと思う。

このように、萌えは恋愛が日本的に歪み、日本的な対象への絶対化が起こり、そうする人々の日本的な集まりによって成長する、いかにも日本的な現象であることがわかる。この背景を無視して萌えを単なる絵、つまり芸術として捕らえるのは間違いである。海外に対する芸術面での輸出が不調なのはそのためである。一方恋愛による自我の維持も紙が担っていた時代に比べればきわめて脆弱であり、海外でも恋愛弱者が生まれつつある。それに対して日本のキャラクターは有効であるだろうか。

6.4 情報リテラシーの欠如

「空気」が支配する日本においては、各々の状況に対する意思決定に必要な情報は、空気から得ることを余儀なくされる。裏を返せば、空気さえ読めばその他の情報は必要ない。結果として日本人の情報リテラシー（コンピュータを使えるかではなく情報を主体的に読み解く能力）は諸外国に比べて低い。大手メディアは個人を操る強力なノウハウを持っており、個人はメディアで押し付けられた「個性」を自分の個性として受け入れる。もっとも、最近は後述するように大手メディアの影響力は低くなっている。

7 2ちゃんねる〜もっとも純粋な「日本的」掲示板〜

インターネットが今のような完全な双方向メディアとなるのには時間を要した。個人には情報を発信する手段がなかったのだ。技術者はサイトを開設して HTML でソースを書けばよい。しかし、それは個人には敷居が高すぎるし、肝心の情報以外のことに気を取られてしまう。掲示板やチャットはどうだろう。これは規模が限定されているのでどうしても多くの人間に発信することができない。そんな中、日本ではアンダーグラウンドの掲示板が人口爆発の問題を抱えていた。単純な BBS では多くの書き込みは更新さえすれば流されてしまう。しかし、日本ではスレッドフロート式掲示板という独自のシステムが誕生し、人口その他の問題を解決した。このシステムはすべての話題をカバーし、アングラな人間に限らず 1 千万人の大衆さえ飲みこむことができた。後述する「個」のメディアではなく「集団」のメディアである。こうして日本のネットコミュニティは独自の道を歩むことになる。

7.1 自由か、カオスか

巨大掲示板「2ちゃんねる」は様々な視点で見られている。古くは「便所の落書き」「アングラ」からはじまり主にその自由度や「カオス」であることをよく強調される。確かに、そこには事実上一切の規制は無く、匿名性から自分の言いたいことが言える環境にある。大学入試センター試験で問題が流出したり、「祭り」のときの公然とした個人情報の流出など、厳重に守られている情報も 2ちゃんねるではすぐに手に入る。ただ叩き合っているかと思えばそうでもなく、相当高度な建設的議論やまとめサイトなどの情報の集積も見られる。ここは自由の楽園であり、大手メディアや政治の支配から逃れる力として働く可能性を秘めている。

7.2 顕在化した日本的思考

ここで実際の書き込みを見てみる。みんな一丸となって韓国を叩いている。選挙のときはみんな一丸となって自民党を応援している。大手メディアの支配から逃れた自由な言論がそこにある。さて、韓国に対する好意的な書き込みや、相当数を占める民主党支持の書き込みはどこへ行ったのだろうか。実は、それらの意見は事実上圧殺されている。2ちゃんねる全体の雰囲気を見ると、韓流ファンが多い（ババァの集まり）既婚女性板などを除けば 2ちゃんねる全体で韓国擁護の書き込みは「韓国人」「ホロン部」扱いされ相手にされない。また、民主党支持に至っては、ネットでは自民党支持という世論のようなものが形成されており、「売国奴」扱いされる。

そう、2 ちゃんねるには事実上自由は無く、カオスの対極に位置する秩序さえできているのだ。特に「祭り」の時には多くの人間が自発的に参加して一つの目的に向かってまい進するようになる。これはよく「2 ちゃんねるの力」と表現される。しかし、明確な指揮系統やリーダーが存在しないはずの 2 ちゃんねるでなぜこのようなことが起きるのだろうか。

これには先ほどの「空気」が関与していることが考えられる。確かに「空気」の力があればこのような書き込みの一極化は自然になされる。その中で空気という言葉自体も盛り返し、「空気読め」あるいは「空気嫁」が 2 ちゃんねる辞典に載るなど一種のネットスラングとしての地位を確立した。実際、スレッドにおいて意見が対立すると、多数を占める（と思われる）ベクトルを持った書き込みをする人間が「空気嫁」と頻繁に書き込み、結局反対する書き込みは消滅する。前述のように指揮系統が存在しない 2 ちゃんねるでは書き込みを支配するのは「空気」であることに他ならない。政党などの特定のイデオロギーも、スレッドによって支配的な考え方が右翼、左翼両方にまたがることを考えるとどうやら圧殺されているようである。日本人は自由な価値観を得た瞬間に自由な価値観を捨てる。

こう考えると 2 ちゃんねるにおける情報の操作も適切な手法を用いれば十分可能であることが示唆される。たとえばメディアミックスで巨大な収益を上げた電車男も、事実であると鵜呑みにするのは難しい。しかし、偽であることを立証することはできないし、実際に電車男以外の人間も参加している。また、ソニーのプレイステーション 2 を擁護するソニー関係者がいて、それがばれた途端にソニー擁護の書き込みが減った（そして、ソニーを叩く空気が支配的になった）こともある。

以上より、2 ちゃんねるは現実での日本人の行動となんら変わることはないことがわかる。ここでは「空気」の呪縛から逃れることはできないどころかむしろ現実世界より強力な空気が支配している。2 ちゃんねるは以前は確かにアングラであり、カオスであった。匿名であることがプラスに働き、本当の意見や考えがスレッドを彩っていた。しかし現在は空気に支配され、差しさわりの無い会話が増え本当の議論の少ないものになった。端的に言えば「つまらなくなった」。2 ちゃんねる初期のコテハン「ワイン比呂有紀」は、文章「ネットバカ一代」においてこう記している。「あの野郎達」はいったい何を指すのだろうか。

いつだってそうだ。流行こそ癌そのもの。
俺のお気に入り、パンク・ロックをファッションに変え、坂本龍一を一時的な流行りものにしてしまい、匿名掲示板を聞きたくもねぇ私の現状報告の場と、団地妻の井戸端会議以下の幼児雑談に変えた、何処からともなくワンサカ集まって来るいつものあの野郎達だ。身近なお祭り騒ぎには参加せずにはいられず、多様な価値観だの諸行無常をぬかすわりには何の精神も思想も持たず、それまで守られてきた大事な大事な宝をあっさりとぶち壊し、ただ目の前にあるバカ騒ぎを刹那的に楽しむだけの糞野郎共。

最後にやつらは言う。「未だにそんな事やってるの？」

7.3 VIP の成功と崩壊

8 Web2.0〜「個のメディア」という括り〜

去る 2001 年 9 月 11 日、米国で忌まわしい同時多発テロが発生した。世界の安定を揺るがす事件はこれまでも多数あったが、今回の場合は少し状況が違った。完全な個人が自分の意見を発信し、議論する地盤がインターネットにより整っていたのだ。当然のように、世界中の人々が一切の装飾、虚偽なしの主張と議論を求めた。かの 2 ちゃんねるも例外ではない。8 月 26 日の閉鎖危機を乗り越えた 2 ちゃんねるでは、珍しく本音の書き込みであふれ返った。本音が「空気」に打ち勝ったのだ。本当の不安と恐怖に満ちた「また戦争になるの?怖いよ…」の発言を私は忘れないだろう。では、実際に被害を受けた米国ではどうだろう。実は当時海外では統一的に主張、議論する場が無く（当時、日本の 2 ちゃんねる、海外の slashdot と言われていた。規模の違いは明らかだろう）、限

定的な人数のフォーラムか 1 対 1 のメールで議論するよりほか無かった。しかしそれとほぼ同時期、一つの Web アプリケーションが産声を上げていた。Blog の代表格、Movable Type である。人々は飛びついた。一つの世界が形成された瞬間である。

近年インターネット上に個々の人間やサイト、ノードが主体の新しいアプリケーションが話題を呼んでいる。Blog をはじめとして、RSS やソーシャルネットワーク、Wiki、アフィリエイト、P2P などである。これらは総称して「Web2.0」と呼ばれ、新しい付加価値を生むという確証のない期待を受けている。付加価値はともかく、情報伝達の手段としてのインターネットは Web2.0 によって大きく変化した。ここでは Blog とソーシャルネットワークを見てみる。

8.1 輸入された Blogosphere

先に述べた経緯で誕生した Blog は、世界を揺るがすニュースから日常の話題まで、「ハッキングから夜のおかずまで」カバーした 2 ちゃんねると同様の広い言論空間を構築した。その中で一部の先進的な（海外かぶれの）日本のネットユーザーが日本のネット空間に Blog を導入し始めた。この流れは今やコアなユーザーにとどまらず、元からあった日記文化を取り込んで大衆的なものになった。Blog を利用して名を売る人間も増えた。

Blog の中でも重要なファクターは、Trackback である。これは人の Blog と関連のある情報を自分の Blog に書いた場合、人の Blog にそれを書いたことを通知するシステムである。これによりコミュニケーションは「人のサイトに赴く」ではなく完全に対等となり、活発な議論が行われている Comment 欄もあり Blog は単体では情報配信媒体というよりコミュニケーションツールとして成功した。それに加えて RSS の積極的な利用も Blog を活性化させた。RSS は Blog などのサイトの記事の要約を配信するシステムで、これにより情報の受け手としてのユーザーは大手メディアと同じように個人の Blog を「購読」することができる。最近では Google や MS なども RSS により人間関係と嗜好を把握するため本腰を入れており、昔は大手の Yahoo などであったポータルを RSS で個人向けにカスタマイズできるようなサービスを行っている。結果的に海外では Blogosphere という巨大な Blog の集合体が生まれた。

さて、どうもコミュニケーションツールとしての Blog は日本では少し勝手が違う。海外と同様のコミュニティと、村社会的なコミュニティの 2 種類が存在するのだ。Trackback や Comment で作られるリンク構造は相互監視の手段として村社会をより強固なものにすることにも非常に役立つ。本音が何一つないクソ Blog も多い。中には批判的な Trackback を「非常識」とする例もあるらしい。自由であるべきだという論とどちらが非常識なのだろうか。そのような Blog は往々にしてネットワーク的に閉じており、外部からの Comment に対して冷たい態度を取る。結局村社会が形成されてしまうのだ。

8.2 反発した 2 ちゃんねる、崩壊した 2 ちゃんねる系 Blog

村社会的なコミュニティと本当に自由な Blog が出会うとどうなるか。それを 2 ちゃんねるで有名になった山本一郎@切込隊長の Blog が表している。2 ちゃんねる社会で彼を批判する「空気」が多くなると、Comment 欄が 2 ちゃんねる化する現象が起こった。もはやここでは自由な討論などできない。個人の Blog なら排除可能だが、2 ちゃんねるは巨大なので手の付けようがない。Blog は 2 ちゃんねるでは有益なソースを除きどちらかというと「晒しあげ」の対象になる。「こんな奴見つけた」から始まりそこに突撃し、Comment 欄に良識ある人間として書き込む。このときだけ 2 ちゃんねらーは独特の用語をやめる。あくまでも「普通の」人として振舞うのだ。もしかしたらこれが 2 ちゃんねらーの正体なのかもしれない。2 ちゃんねるで唯一流行する Blog はスレッドを紹介するものである。完全に内輪で閉じているのは言うまでもない。

8.3 mixi 八分

「一人で生きているわけじゃない」そういう日本人は多い。これは空気を読んで思いやりを持つよう諭すことを含意しているが、言葉自体の意味はなんら間違ってはいない。人は一人では生きられない。それゆえ周囲の人

間関係もその人間を規定する要素である。本当の個人メディアになるためには周囲とのリンクが存在するべきであるというはBlogが既に証明している。そのリンクをもっとはっきりと示せばより良い空間になるだろう。そんなサービスがソーシャルネットワーキングサービス（SNS）である。これは自分の友人を友達リストとして自分のプロフィールに加えることができるもので、もちろん掲示板などもある。個人をより保証するために紹介制にしているものもある。海外ではFriendSterを皮切りとしてGoogleに買収されたOrkutなどが流行した。日本では少し前はGREE、現在はmixiがユーザーを伸ばしている。

mixiはよく2ちゃんねらーに馴れ合いと称される（お前らもだろ!)。だが、画期的な個人メディアであるSNSではさすがに日本人も存分に個性を出しているだろう。そのはずが実際に村社会的な馴れ合いの空気が横行している。mixiの日記はBlogよりさらに薄まっている。誰が自分のスペースを訪れたかを見ることができる「足あと」で相互監視がさらに強まっている（Blogでもアクセスログを監視している人間がいるが)。最悪なのが友達リストであるマイミクシィを特定の集団から一気にはずされる「mixi八分」である。これは明確な形で見ることができる。そうでなくてもマイミクシィの扱いには注意しないといけない。外すことに抵抗を感じていろいろ突っかかってくる人がいるから外せない。追加も「親しい人でないとだめ」ばかりで、追加すると親近感を感じて突っかかってくる人が多い。まあナンパはかまわないと思う。結局mixiでも村社会が形成され、明示的に人間関係が見えるために状況をさらに悪化させている。SNSは日本人を変えなかった。もちろんmixiでは個性を出している人も多いということを付け加えておく。コミュニティでは激論が交わされ、頻繁なオフ会でさらにコミュニケーションを重ねる。内輪のみのコミュニティと馴れ合うためのオフを尻目に。

9 Web2.0は日本人を変えるか

以上で見てきた2つの「個」のメディアは、多くの人間を本当のコミュニケーションに誘う場となると同時に、旧来の村社会をさらに助長する要素としても働いている。結果として2つの種類の人間の間の溝はさらに深まっている。幸いにして、BlogやSNSでは明確な仕切りを設けることができる。一方はさらにコミュニケーションに精を出し、もう一方はさらに馴れ合う。この二重構造は日本特有の問題である。日本で本当のコミュニケーションと多様な価値観は根付くのだろうか。

9.1 モヒカン族というミーム

それに関連して、最近のBlog界で「モヒカン族」という言葉を聞く。モヒカン族は技術者から発生したもので、個人ではなく発言にのみ注目する、情報を共有する、馴れ合いよりも殺伐とした議論を好み、他人の間違いを修正することを推奨する、たくさんの人がよくなる方法が良しとする、互いの違いを確認するという「モヒカン宣言」に基づく。これは旧来のコミュニティを「ムラ社会」（村社会とはニュアンスが違う）と呼び、自分たちをその対極の「モヒカン族」と位置づけている。

これは対極であるが、よいコミュニケーションとも言えない。殺伐としているだけでは、コミュニケーションの重要な機能である調整はすぐに空気によって論理のない調整がなされる村社会と同様に成功はしないだろう。また、個人の不在は発言の源泉の理解を妨げ、真意を伝えることを妨げることになる。技術だけならよいが他のコミュニケーションにはどうも不向きである気がしてならない。

もっとも悪いのは、モヒカン族という言葉が認知されることでモヒカン族のコミュニティ自体が村社会の性質を帯び始めていることである。既に「あいつはモヒカン族である」と言う類の踏み絵が用意され、モヒカン族同士の馴れ合い的な連帯もみられる。斧を持ったモヒカンは許されるが、斧を持ったロンゲは許さないのだ。そもそも構成員が村社会出身だから仕方ないのかもしれないが。最後は単なる一つの村社会になり消滅するだろう。

9.2 結び〜EPIC 2014の描く未来〜

EPIC 2014という未来予測Flashが最近のBlog界で話題になった。これは現在のWeb2.0の方向の最終形である、情報の発信も受信も究極的にパーソナライズされたシステムをGoogleとamazonが構築するというもの

である。ユーザーはあらゆる種類の情報を発信し、嗜好から人間関係まであらゆる情報を元にプログラムがユーザー向けに記事を書く。Flash の最後にこのシステムの 2 つの未来が示されている。その日本語訳はこうである。

私たちのすべては多くの編集者を購読するようになる：EPIC では、彼らが選んだ記事を好きなように組み合わせることができる。最高の状態では、EPIC は、見識のある読者に向けて編集された、より深く、より幅広く、より詳細にこだわった世界の要約といえる。

しかし、最悪の場合、多くの人にとって、EPIC はささいな情報の単なる寄せ集めになる。その多くが真実ではなく、狭く浅く、そして扇情的な内容となる。

しかし、EPIC は、私たちが求めたものであり、選んだものである。そして、その商業的な成功は、報道倫理のためのメディアと民主主義をめぐる議論が起こる前に実現した。

この対比は日本での村社会と個性の対比にそのまま適用できる。われわれはどの道を選ぶのだろうか。恐らく現在の二極化された状況では、それぞれに合わせた情報が配信されるだろう。
　ここまでで、新しいパラダイムで変わる日本人と変わらない日本人を見てきた。海外で生まれたアプリケーションが日本的に利用される。まだ日本で本物の言論は生まれない。私の場合、もともと村社会に無理に迎合しようとして失敗し、モヒカン族的になった後に個性を出し、独自の価値観を生かすことを選んだ。それには村と個性両方の人間に実際に出会い、会話を交わしたことがかかわっている。現在のネットの情報では現実と匹敵するコミュニケーションはできない。現実世界での対話なくしてネットでの変化はないと思う。

第 IV 部
FreeBSD6.0 によるデスクトップ環境

(著者) 0511106 村松雄介

概要

この文章は,BSD [12] や Linux [13] はサーバ用途でしか使ったことの無い筆者が, FreeBSD6.0 にてデスクトップ環境を構築した際の方法をまとめたものです.

読者の対象は*nix [14] に興味を持つ平均的な J 科 1 年[15] を想定し, その読者を BSD に染める目的で記述されています. そのため, 脚注を大量に記述してあります. この文章を参照する場合, 図が非常に少ないため, 事前に FreeBSD のインストール作業を体験するか, 実際にインストール作業を行いながら参照するとより効果的でしょう.

10 動機, 目的

筆者は電気通信大学へ入学するまでの間, デスクトップ環境として Windows, サーバとして Gentoo Linux [16] と Windows を併用していました.

平成 17 年に電気通信大学電気通信学部情報工学科に入学したことにより, 次に挙げるような要求が発生したため, それまで Windows にて構築していたデスクトップ環境を, FreeBSD でリプレースしました.

- 純粋な*nix プログラミング環境[17]
- より高度な*nix リテラシー
- エンターテイメント環境と勉強のための環境の分離[18]
- 信頼性の低い Windows からメールなどの重要データの隔離[19]
- その他雑他な理由

上記の要求へのソリューションとして FreeBSD6.0 を選択した理由は, 筆者の所属する MMA に BSD ユーザーが多いこと, BSD の開発思想[20] に共感を覚えたためからです.

11 FreeBSD とは

FreeBSD は一般的な PC/AT 互換機で動作する UNIX 互換の OS です. インターネット上のサーバとして広く用いられており, Yahoo や楽天, かの Microsoft でさえもウェブサーバに FreeBSD を採用しています. また, パワーユーザーのデスクトップ環境としても使用されています.

FreeBSD はソースコードも含めて無償で公開されており, 世界中のボランティアによって開発されています.

---

[12] unix ライクなオープンソース OS の一種.
[13] unix ライクなオープンソース OS の一種.
[14] Unix の商標は The open group が持ってるので, 使用料を払っていない BSD や Linux では使うことが出来ない.
[15] *nix リテラシーはあるが自分で環境を作ったことはない, 程度.
[16] Linux ディストリビューションの一種. Linux ディストリビューションとは, OS の心臓部である Linux kernel とその他のプログラムをまとめたものです.
[17] Windows 上でも cygwin などで擬似的に構築することは可能だが色々面倒であるためです.
[18] 筆者はできた人間ではないので, レポートを打っている最中に誘惑に負けることが多々あるためです.
[19] バックアップは行っているのでデータは失わないが, 復旧作業が面倒であるので.
[20] BSD は信頼性と安定が第一.Linux は速さを優先する風潮がある.

12 ハードウェア

今回の環境は sgi 1200 に構築しました. sgi 1200 は Silicon Graphics が製造した x86 アーキテクチャ[21] のラック型[22] サーバです.

スペックは以下の通りです.

- Pentium III 700MHz Dual
- PC100 ECC Registered SDRAM 1GBytes
- Barracuda 7200rpm 18GBytes RAID0
- DAC960PTL

デスクトップ環境として運用する関係上, 命の次に大事なソースコードなどを保存するため, DAC960PTL による RAID1 [23] で HDD を冗長化しています.

RAID1 の構築は, 起動時に AltR を押すことにより, DAC960PTL の EzAssist と呼ばれる BIOS に入り設定しました. RAID の構成が行われていないディスクを HDD マウンタに挿入し EzAssist を起動すると, RAID の構成の Wizerd が表示されます. 今回はそれに従って RAID1 を構築しました.

なお, 我が家のホストはすべて UPS [24] で停電から保護されているため, HDD への書き込みをライトスルー[25] からライトバック[26] に変更しました.

13 FreeBSD6.0 のインストール

13.1 インストールメディアの入手

FreeBSD6.0 をインストールするためにはインストール用の CD を用意する必要があります[27]. 筆者は BitTorrent [28] で入手しました.

iso イメージをダウンロードしたら CD-R に書き込みを行います. Disc1 と Disc2 の書き込みを行ってください[29].

13.2 インストール準備

Disc1 を CD-ROM ドライブに挿入し, FreeBSD6.0 を CD からブートしました. 最初に bootloader が表示された後, sysinstall [30] が表示されます.

usage を斜め読みした後, standerd を選びインストール作業を開始します.

ここからの作業は, 最後に確認のダイアログが表示されるまで HDD に反映されることは一切ありません. 間違えても CtrlAltDel でトップメニューに戻ってやり直せるので, 肩の力を抜いて作業してください.

最初に DiskPartition Editor でスライス[31] を切ります. 今回は FreeBSD のみをインストールするのですべての領域を割り当てました. a キーを押すと自動的に割り当てを行ってくれます. q で終了です.

---

[21] Intel i386 系の 32bitCPU を搭載した計算機. 有名どころでは Pentium シリーズや Athlon シリーズ.
[22] 19 インチラック. データセンタなどで大量のサーバを並べるときに用いる.
[23] 同じデータを 2 つのディスクに書き込みます. 他方が壊れてもデータは失われない.
[24] 無停電電源装置. 停電時にバッテリーから電力を供給する.
[25] キャッシュメモリに溜め込まずディスクに直接書き込む.
[26] キャッシュメモリに溜め込んだらすぐ他の処理をする. まだディスクに書き込まれていないときにトラブルが発生するとそのデータは失われる.
[27] 入手方法は http://www.freebsd.org/を参照してください.
[28] BitTorrent クライアントは有志の手により複数開発されており, Windows 上では BitComet などがあります.
[29] 用途が限定されますが,DeepBurner などがフリーで使えます.
[30] インストーラー兼システム管理ソフトです.
[31] Windows でいうパーティションです. BSD でいうパーティションとは似て非なるものです.

表 1: ディストリビューションの内容

ディストリビューション	内容
User	基本アプリ, ドキュメント
Developer	基本アプリ, ドキュメント, ソースコード
Kern-Developer	基本アプリ, ドキュメント, カーネルソースコード
X-User	User, X.Org
X-Developer	Developer, X.Org
X-Kern-Developer	Kern-Developer, X.Org
All	すべて
Custom	手動選択

次に BootLoader をどうするか聞いてきます. BootMgr な BootLoader を選ぶと複数の OS を選んで起動することが可能です. ここでは FreeBSD のみを使うため, standard の BootLoader を MBR に書き込みます.

DiskLabel Editor でパーティション[32] を切っていきます. FreeBSD5.4 までは自動設定の値が時代遅れで自分で決定する必要がありましたが, FreeBSD6.0 では改善され, a キーを押して自動設定される値をそのまま用いても特に問題がないようになっています. サーバ用途では手動で書き換えていますが, 今回はデスクトップ用途なので特に考えず a キーを押し自動設定されたものをそのまま使いました. q キーで抜けます. 参考までに自動設定された値を以下に示します.

- / 512MB
- swap 2GB
- /var 2GB
- /tmp 512MB
- /usr 12.5GB

次はディストリビューション[33] の選択です. 各ディストリビューションで導入されるファイルを表 1 に示します.

選択後に Custom を選択することにより, どのようなものがインストールされるかを見ることが可能です.

ソースコード[34] は要らないと考える方も多いでしょうが, FreeBSD の場合はソースコードからシステムをすべてコンパイルしなおせるので, 入れることを強く勧めます. 筆者は, X.Org は後で ports からインストールすることとし, Developer を選択, Exit を選択し Enter を押しました.

ディストリビューションの選択中に[35] ports [36] を入れるか聞いてきます. Yes を選択しました.

最後にインストールメディアを選択します. FreeBSD ではネットワークから FTP などでファイルを取得し, インストールを行うことが可能です. 今回はフル構成の CD を用意し CD からファイルを読み込むので CD/DVD を選択しました.

[32]スライスをさらに分割するもの.Windows のパーティションとは違います. 1 階層ぶん内側の概念です.
[33]FreeBSD でのディストリビューションは, どのソフト (X.Org など), データ群 (ソースなど) を入れるかを指します.
[34]FreeBSD そのもののソースコードです. カーネルソースは BSD カーネルのソースコードです.
[35]かなり不可解なタイミングです.
[36]我らが BSD が誇るとっても素敵なパッケージ管理システムです. ありとあらゆるプログラムをソースコードからコンパイルしてインストールすることができます. 一度はまると病み付きです.

表 2: ネットワーク設定

項目	設定内容
Host	long.muraexe.sytes.net
Domain	muraexe.sytes.net
IPv4 Gateway	192.168.1.1
Name server	192.168.1.1
IPv4 Address	192.168.1.2
Netmask	255.255.255.0

13.3 インストールの実行

Last Chance!から始まるダイアログが表示されます. Yes を選択し, 実際に HDD へパーティションの書き込みやファイルの書き込みを行いました. このダイアログで Yes を選択するまでは HDD には一切変更が加えられないので, やり直しをすることが可能です.

数分から数十分でインストールは完了し, Congratulations!から始まるダイアログが表示されます.

13.4 インストール後の設定

configure any Ethernet のダイアログでは, Yes を選択し, ネットワーク接続の設定を行いました.

ネットワークデバイスの一覧から Ethernet card と思われる選択肢を選び, OK を選択します. sgi 1200 では Intel の Ethenet が内蔵されているので, fxp0 [37] を選択しました.

IPv6 [38] configuration では No を選択しました.

筆者の自宅の,sgi 1200 が接続されているサブネットでは, 恒常的に設置されている PC に対して DHCP [39] での IP address の割り当ては行っていないため, DHCP configuration では No を選択しました.

Network Configuration の画面で, ネットワークの設定を行います. 設定内容を表 2 に示します.

Host, Domain に関しては, 分からなければ[40] freebsd, freebsd.localhost とでも入力しておきましょう. 入力完了後 OK を選択し設定画面を抜けました. interface up right now では Yes を選択し, ネットワークインターフェイスを LinkUp [41] させました.

network gateway では No を選択しました[42].

configure inet [43] では No を選択しました. 反射的に No を選択しましたが, デスクトップ環境の場合 inetd を導入することで特に不具合は出ないので, この選択でよかったのか微妙に悩んでいます.

enable SSH [44] login では Yes を選択しました.

anonymous FTP [45] では当然のごとく No を選択しました.

NFS server と NFS client [46] では No を選択しました. 今後他のホストとディレクトリの共有を行う予定がありますが, rc.conf をちょこちょこと書き換えるだけで済むので,No を選択しました.

[37]搭載されているチップによって名称が変わります.

[38]現在インターネットで利用されているのは IPv4 で, その次世代版です. UEC のネットワークは IPv6 も使用されていますが, 一般家庭で使用されることはあまりありません.

[39]DHCP は自動的に IP address を割り当てる方法です. ブロードバンドルータにこの機能がついている場合が多いです.

[40]分かる人はそれなりの知識はあるはずですよね.

[41]Ethernet を接続された状態にすること. Windows の場合は勝手に LinkUp するので意識することはあまりありません.

[42]このホストをルータとして動作させるかの設定です.

[43]inetd はネットワーク接続を受け付けるデーモンです. inetd は接続要求を受け取ると, 該当するデーモンを起動し接続を渡します.

[44]Secure SHell は, 外部からホストにログインするための方法です. Windows からは putty などを利用して接続できます.

[45]FTP を誰にでも公開するという設定です. 有効にするなら相当慎重な運用を行う必要があります.

[46]Network File SystemWindows でいう共有フォルダみたいなものです. Unix 同士のディレクトリの共有で用いられています.

customize your system console では No を選択しました. 筆者は英語キーボード[47] を使用しているため No を選択しましたが, 日本語配列キーボードを使用している方は, ここで Yes を選択し, Keymap の設定を行うとよいでしょう.

set this machine's time zone では Yes を選択し, TimeZone の設定を行いました. このホストは日本で使用するので, set to UTC [48] では No, Time Zoen Selector では Asia,Japan を選択しました.

enable Linux binary compatibility [49] では, あとで ports からインストールを行う予定であるため No を選択しました.

have a PS/2 [50] , serial, or bus mouse では Yes を選択し, マウスの設定を行いました. Enable を選択し, マウスカーソルが動くことを確認した後, Yes を押してダイアログを閉じ OK で設定画面を抜けました.

browse the (packages [51]) collection では No を選択しました. しかし, 後述しますが, ここで cvsup-without-gui を入れるべきでした.

add any initial user では Yes を選択し, 普段使うための[52] 新しいユーザーアカウントを作成しました. User を選択し Enter を押します. 入力内容を以下に示します.

- Login ID には適当な 16 文字以内のユーザー名を入力します.
- UID はデフォルトの値をそのまま指定してください.
- Group は空欄のままにしておきます.
- Password にはパスワードを入力します.
- Full name ユーザーのフルネームを入力します.
- Member groups は,wheel と入力します.
- Home directory はそのままの値を使用します.
- Login shell には/bin/tcsh と入力します.

Login ID は*nix の伝統としては 8 文字にすると良いでしょう. パスワードはできる限り複雑なものを使用してください. 何らかの単語で 8 文字以下とかは論外です. Full name は他のユーザから参照することが可能です. wheel とは管理者のグループです. Home directory [53] を変更する必要は特にありません. Login shell は tcsh か bash [54] をお勧めします.

表に各設定値の例[55] を示します.

OK を選択し, ユーザの追加が完了したところで Exit を選択し画面を抜けます.

続いて root のパスワードを設定します. 10 から 20 文字ぐらいのランダムなパスワードを設定しました. このパスワードが悪意ある攻撃者に知られてしまうと大変危険[56] なので, 複雑かつ長いものにし, 紙のメモなどは残さないようにします[57].

最後に, visit the general configuration menu と表示されるので, No を選択しました.

---

[47] Happy Hacking Keyboard 英字配列です. Unix を使うなら非常にお勧めします. Windows で使用すると悲しくなる可能性が高いです. 日本語配列版は邪道です.

[48] Coordinated Universal Time です. グリニッジ標準時みたいなものです.

[49] FreeBSD では,Linux の実行ファイルを直接実行することが可能で, その機能です.

[50] キーボードとマウスを繋ぐ丸いコネクタです.

[51] FreeBSD で用意されているバイナリパッケージ配布の仕組みです. ports の方が素敵なのであまり使われません.

[52] 管理者アカウント root はコマンド一発でシステムを破壊できる強力なアカウントなので常用するのは大変危険です. スーパーユーザとも呼びます.

[53] ユーザの作成したファイルはすべてここに保存されます. Windows でいうマイドキュメントやデスクトップ相当です.

[54] BSD 的には tcsh です. bash は Linux のみが主に使っています. bash に変更するようなパワーユーザはこの文章を読む必要はあまりなさそうです.

[55] 当然パスワードなどは架空の値です.

[56] 攻撃の踏み台にされ, ある日突然, 京都府警の方があなたの自宅にやってくるかもしれません.

[57] 暗記するまではパスワードの一部を欠落させたり入れ替えたメモを用意しておくと良いでしょう.

表 3: Add a new user

項目	値
Login ID	hoge
UID	1001
Group	空欄
Password	pXh0W4df6m
Full name	Hogeo PIYOTA
Member groups	wheel
Home directory	/home/hoge
Login shell	/bin/tcsh

これでインストールは完了です. Exit install を Tab キーで選択し Enter を押しましょう. ダイアログの内容にしたがって CD を抜き, ホストにインストールした FreeBSD が起動します.

14 FreeBSD の設定

無事 FreeBSD6.0 が起動したので, FreeBSD 自体の設定を行っていきます.

14.1 初回起動時の設定

起動中はカーネル[58] の吐き出すメッセージがずらずらと表示されます.

途中, kern.random.sys.seeded:と表示され停止します. ここでは Enter を押し先に進みます. ssh などで使用されるフィンガープリント[59] を作成するものです.

14.2 CVSup

最初に CD に書き込まれている内容は少々古い[60] ので, CVSup [61] を導入し, ソースコードを最新のものに更新しました.

ports [62] からインストールしようと考え次のようにコマンド[63] を入力しました.

```
% cd /usr/ports/net/cvsup-without-gui
% su
# make install
```

ところが, インストールに非常に時間がかかります. 表示されているメッセージを良く見ると, Modula3 [64] をダウンロードしているようです[65]. CtrlC を入力し中断し packages からのインストールを行うことにしました.

[58] FreeBSD の OS の中枢部分です. 各種デバイスの管理やメモリの管理など OS の機能のほとんどが含まれています.

[59] ホストを識別するための一意な数字. 成りすましを防ぐ目的で利用される.

[60] FreeBSD 的にはです. 数ヶ月程度です.

[61] FreeBSD でソースコードの配布に使われている技術. 差分だけを送るのでネットワークにやさしい. 参照先が複数あり更新頻度や運用方針が違う. current などは日単位で更新されている.

[62] 詳細は後で説明します.

[63] %から始まる行は tcsh 系の一般ユーザで, bash な一般ユーザは$, #から始まる行は root で入力します. su コマンドで一時的に root になることができます. exit で元に戻ります.

[64] とてもマイナーなプログラミング言語. CVSup 以外に使っているのを見たことが無いです.

[65] 依存関係がある場合自動的にインストールしてくれるのも ports の便利なところです.

packages は FreeBSD におけるバイナリパッケージ[66] 配布システムです. コンパイル済みなので Modula3 の環境を構築しなくても CVSup をインストールすることが可能です.

packages は pkg_add コマンドなどを使って操作することも可能ですが, たまにしか使わないものなので, 便利な sysinstall から packages を利用することにしました.

```
# sysinstall
```

Configure に入り packages を選びました. できるだけ新しいものが欲しいので Installation Media は FTP を選択しました. サーバは Japan の中から番号の大きなものを適当に選びます. net カテゴリの中の cvsup-without-gui をスペースキーで選択し, Tab キーで OK を選択し Enter, インストールされるパッケージの内容を確認し Enter を押し, パッケージがインストールされるまで数分待ちました.

今回は問題なく CVSup の導入が済みました.

続いて,/etc/make.conf [67] を書き換えて CVSup の設定とコンパイルオプションの設定を行いました.

まず始めに, サンプルファイル[68] をコピーしました.

```
# cd /etc
# cp /usr/share/examples/etc/make.conf ./
```

次に ee [69] コマンドで make.conf を編集しました.

```
# ee make.conf
```

次の行が有効[70] になるよう編集しました.

```
CPUTYPE=i686
CFLAGS= -O -pipe
COPTFLAGS= -O -pipe
SUP_UPDATE=     yes
SUP=            /usr/local/bin/cvsup
SUPFLAGS=       -g -L 2
SUPHOST=        cvsup6.jp.FreeBSD.org
SUPFILE=        /usr/share/examples/cvsup/standard-supfile
PORTSSUPFILE=   /usr/share/examples/cvsup/ports-supfile
```

pentium3 と指定しても良かったのですが, なんとなく i686 [71] にしました. CFLAGS は-O [72] と-pipe [73] を指定. SUP で始まる行は make update で CVSup が自動的に行われるようにする設定です.

設定が完了したら/usr/src [74] に移動し, ソースコードを最新の状態に更新しました.

---

[66]既にコンパイルされた実行ファイル.

[67]FreeBSD システム全体の構築, 特にコンパイルについての設定ファイルです. /etc は FreeBSD そのものに関する設定ファイルが大量においてあります.

[68]/usr の examples というディレクトリにあることが多いです. 一般的には/usr/share/examples

[69]FreeBSD での Vi 以外の標準エディタです. 筆者は emacs 派ですがまだ emacs 入れてないので.

[70]一般的に*nix の設定ファイルでは, #以降の部分はコメントとして無効化されます.

[71]PentiumPro 以降の一般的な x86 系列の CPU です.

[72]Optimeze 最適化オプション. -O や-O2 はあまり過激な最適化は行いませんが, -O3 はループ展開など強力な最適化を行うので動作がおかしくなったりします. ssh できなくなるなど.

[73]コンパイルが少し速くなります.

[74]/usr/src は FreeBSD に関係するソースコードが格納されています.

```
# cd /usr/src
# make update
```

基本的に長い時間, 特に初回はかなりの時間がかかるので, 就寝前に実行しました.

14.3 カーネルの再構築

sgi 1200 は SMP [75] なので, カーネルをコンパイルしなおして SMP 対応にしました.

```
# cd /usr/src/sys/i386/conf
# config SMP
# cd /usr/src/sys/i386/compile/SMP
# make cleandepend
# make depend
# make
# make install
```

筆者は普段, カーネルの config ファイル GENERIC をコピーし, オプションを弄ってから[76] 再構築を行うのですが, 今回は簡単のためにサンプルの SMP 対応 config ファイルをそのまま使いました.

各段階でエラーが出ていないか確認してください. もしエラーが出ている場合は絶対に make install を行わないようにしてください. 最悪の場合起動しなくなります[77].

ここで, 正常に起動できるカーネルをバックアップするのを忘れていたことに気が付き, 標準のカーネルのバックアップを行いました.

```
# cd /boot
# cp kernel.old ./kernel.GEN
```

新しいカーネルを make install すると, 古いカーネルは kernel.old にリネームされます. make install の 2 回目以降では kernel.old が上書きされてしまうので, 標準の GENERIC カーネルを保存しておくことが推奨されます.

make install とカーネルのバックアップが完了したら, マシンを再起動[78] しましょう.

```
# shutdown -r now
```

起動中に流れるメッセージに SMP: AP CPU #1 Launched!と表示されれば SMP が動作しています. top コマンド[79] を実行すると CPU0 と CPU1 が違う処理を行っていることが分かると思います.

14.4 ports

14.4.1 準備

次に ports を使えるよう準備を行いました. ports ツリーの更新[80] は完了しているので index と readme を作成しましょう.

[75] Symmetric Multiple Processor 対称型マルチプロセッサ. 普通のマルチ CPU マシンです. 今はやりのデュアルコア CPU なども該当します. SMP 以外は相当マニアックです. NUMA とか.

[76] 余計なものを消すと動作が速くなるなど色々利点があります.

[77] 起動時の 10 秒のカウントからローダーに入って違うカーネルから起動したり,FreeSBIE や Fixit Disc から修復することは可能ですが.

[78] reboot などもありますが shutdown を使用しましょう. 他のコマンドは強制終了っぽい動作をします.

[79] ホストの負荷とプロセスの情報を表示します.

[80] make update です.

```
# make index && make readme
Generating INDEX-6 - please wait..perl: not found
===> arabic/ae_fonts_mono failed
*** Error code 1
perl: not found
===> accessibility/at-spi failed
*** Error code 1
2 errors

**********************************************************************
Before reporting this error, verify that you are running a supported
version of FreeBSD (see http://www.FreeBSD.org/ports/) and that you
have a complete and up-to-date ports collection.  (INDEX builds are
not supported with partial or out-of-date ports collections -- in
particular, if you are using cvsup, you must cvsup the "ports-all"
collection, and have no "refuse" files.)  If that is the case, then
report the failure to ports@FreeBSD.org together with relevant
details of your ports configuration (including FreeBSD version,
your architecture, your environment, and your /etc/make.conf
settings, especially compiler flags and WITH/WITHOUT settings).

Note: the latest pre-generated version of INDEX may be fetched
automatically with "make fetchindex".
**********************************************************************

*** Error code 1

Stop in /usr/ports.
*** Error code 1

Stop in /usr/ports.
# make update
# make index
コケる
```

見事にコケました[81]. エラー後の悪あがきも無駄に終わっています. エラーメッセージによると make fetchindex でサーバが作成した index が入手できるらしいので, それでしのぐことにしました. なお, 後述しますがこの問題は解決しました.

```
# make fetchindex
```

これで準備は完了です.

[81] make 系のコマンドは失敗するとこんな感じのエラーを表示します.

14.4.2 使用方法

基本的に,ports は該当ディレクトリに移動して, make install と打ち込むだけですべてやってくれます. 以下に簡単な使い方の例を挙げます.

```
# cd /usr/ports
# make search name="sl" | less
# cd games
# make search name="sl" | less
# cd sl
# ls
# less pkg_descr
# make install
# make clean
# exit
% su
# sl
# make deinstall
# make clean
```

アプリケーションによっては configure が使えることもあります. 重要事項が書かれたテキストファイルがおいてあることもあるので, インストール前によく読みましょう. make に失敗するときは make clean を 2 回実行すると直ったりするようです. また, アプリケーションによっては make 中に設定を要求するものもあるので, make を始めて寝るときなどは注意しましょう.

14.5 mail の foward

FreeBSD では毎日自動的にシステムの状態をチェックし, root に結果をメールしてくれます[82]. そのメールを常用する他のユーザ当てに転送するよう設定しました.

```
ee .forward
```

内容は, 1 行に 1 つユーザ名かメールアドレスを記入します. 筆者はユーザ名のみを記述しました.

```
hoge
```

なお, この作業は X.Org の make 中に行いました.

15 アプリケーションのインストール

15.1 X.Org

15.1.1 make install

このマシンはデスクトップ環境として使うので, X.Org をインストールしました. X.Org は現代的な GUI [83] を提供します.

[82] temp ファイルの削除からパスワードや設定ファイルの整合性のチェックまで色々やってくれます.
[83] Windows 風味といえばいいでしょうか. X のほうが登場は先ですが.

```
# cd /usr/ports/x11/xorg
# make
Ctrl^C
# make clean
# make -j4
```

始め普通に make を行いましたが, -j オプション[84] をつけてみたくなり, 中断して make -j4 を叩きました. しかしコケたので, おとなしく普通に make しました.

```
# make clean && make clean
# make
```

途中で gettext の config が表示されるのですが, ここで誤って examples にチェックを入れて OK を選択してしまいました. 害はないのでそのまま進めても構わなかったのですが, CtrlC を押して中断し, gettext の examples のチェックを外しました.

```
# cd /usr/ports
# make search name=gettext
# cd /usr/ports/devel/gettext
# make clean
# make clean
# make config
examples のチェックを外す
# cd /usr/ports/x11/xorg
# make
```

再度 X.Org の make を行いましたが, ここで, 何度も中断を行ったことにより不整合が発生したようで, エラーを表示して止まってしまいました.

```
===>  Installing for xorg-libraries-6.8.2
===>   Generating temporary packing list
===>  Checking if x11/xorg-libraries already installed
pkg_info: package xorg-libraries-6.8.2 has no origin recorded
===>   xorg-libraries-6.8.2 is already installed
      You may wish to ‘‘make deinstall’’ and install this port again
      by ‘‘make reinstall’’ to upgrade it properly.
      If you really wish to overwrite the old port of x11/xorg-libraries
      without deleting it first, set the variable "FORCE_PKG_REGISTER"
      in your environment or the "make install" command line.
*** Error code 1

Stop in /usr/ports/x11/xorg-libraries.
*** Error code 1

Stop in /usr/ports/x11/xorg.
```

[84] make を複数走らせマルチ CPU の能力を最大限に引き出すことができます. ただしたまにエラーを吐きます.

エラーメッセージでは, アップグレードの時の話をしているので微妙に内容が違いますが, とりあえず make deinstall と make install を行ってみます.

```
# make deinstall
# make reinstall
```

今度は成功しました.

15.1.2 全体の config

筆者は*nix に X.Org を入れて運用した経験がほとんどないため X.Org の設定は少々てこずりました.
インストールした直後の状態でも startx コマンドにより X.Org を使用することが可能ですが, より使いやすくするために設定を行います.

```
# Xorg -configure
# mv /root/xorg.conf.new /etc/X11/xorg.conf
```

Xorg -configure を行うと,X.Org 全体の設定ファイルの雛型が/root/xorg.conf.new に生成されます. これを参考に設定をしました. まず, 画面表示に使うフォントが格納されているフォルダ[85] を追加します.

```
# ee /etc/X11/xorg.conf
```

```
        FontPath     "/usr/X11R6/lib/X11/fonts/TrueType/"
        FontPath     "/usr/X11R6/lib/X11/fonts/local/"
        FontPath     "/usr/X11R6/lib/X11/fonts/bitstream-vera/"
        FontPath     "/usr/X11R6/lib/X11/fonts/URW/"
        FontPath     "/usr/X11R6/lib/X11/fonts/cyrillic/"
```

次にハードウェアの設定を行いました. xorgconfig を使って指示に従い設定を行いました. この設定に入る前に, モニタの解像度, ビデオカードの種類とビデオメモリの量を調べておく必要があります.

```
# xorgconfig -textmode
```

適当な意見ですが, ハードウェアの情報の詳細が分からない場合, 表 4 の値を適用すると大体うまく行きます. depth(色深度, 色数) と解像度の組み合わせはデフォルトの値で OK でしょう.
ただし,xorgconfig の注意書きにもあるように, 誤った設定をするとまれにハードウェアを壊してしまう可能性があります. 表 4 の値は非常に保守的な[86] 値ですが, 適用する際には自己責任でお願いします.

15.1.3 個人の config

先ほどまでの設定はホスト全体に適用される設定でしたが, ここからはユーザごとの設定を行います. X.Org のユーザごとの設定ファイルは /.xinitrc [87] なので, ここに設定を書き込みました.

[85] Windows でいうフォントフォルダといったところでしょうか.
[86] 安全を第一に考えた.
[87] ~/で自分のホームディレクトリの意味です. dot で始まるファイルは隠しファイルなので ls -a で表示しましょう.

表 4: 適当な設定値

項目	ハードの概要	設定値
hsync	15inch CRT	31.5-48.5Hz
	17inch CRT	31.5-64.3Hz
	19inch CRT	31.5-79.0Hz
	14.1-15inch LCD	31.5-48.5Hz
	17-19inch LCD	31.5-64.3Hz
vsync	CRT	50-90
	LCD	50-70
video card	10 年以上前の PC	Generic VGA compatible
	10 年以内の PC	Generic VESA compatible
video memory	10 年以上前の PC	1024K
	10 年以内の PC	2048K
	5 年以内の PC	4096K
default depth	5 年以上前	16bits
	5 年以内	24bits

通常 X.Org を起動するときは xinit を使用します. このコマンドは X.Org の起動後 /.xinitrc を実行し, /.xinitrc の実行が終了すると X.Org も終了されます.

まず, デフォルトの設定ファイル[88] をコピーし, それを編集しました.

```
# find /usr/X11R6 -name "xinitrc"
find: /usr/X11R6/lib/X11/xdm/authdir: Permission denied
/usr/X11R6/lib/X11/xinit/xinitrc
# cd /usr/X11R6/lib/X11/xinit
# cp ./xinitrc ~/.xinitrc
# cd
# ee .xinitrc
```

コピーしたデフォルトのファイルを見ると, start some nice programs 以下の部分で色々と起動しています. twm はウィンドウマネージャーと呼ばれ, ウィンドウの管理などをしています. ためしにコメントアウト[89] して xinit を実行してみるとよいでしょう.

xclock と xterm は文字通りと言った所です. xterm は ASCII 以外の表示で不具合が多いので, mlterm などをインストールし書き換えることをお勧めします.

最後の exec がポイントです. exec は起動されたプログラムが終了するまで先に進みません. xinitrc が最後まで実行されると X.Org が終了するので, このようにしているわけです.

このファイルは日本語の設定でもう一度触れることになります. 今はこのままにして次に進みました.

15.2 ports の index のその後

*nix は他の処理を行っている最中でもとても機敏に動作します. そこで, 前記の X.Org の make 中に/etc を眺めたりしていたのですが, ふと,make fetchindex のみをしたのだから make readme をする必要があると思いつき, 実行してみました.

[88] *nix では, 設定ファイルに関して, とりあえずデフォルトのものが無いか調べ, あったらそれを元に改造するのがセオリーです.
[89] 行頭に#をつけましょう. コメントとして無視されます.

```
# make readme
# make index
# make readme
```

ダメ元で make index を行ったところ, なぜか成功しました. 原因はちょっと良くわかりませんが, うまく行ったので今回はよしとします[90].

また, 気が付いたこととして, 最近の ports は SHA256 [91] でファイルの整合性をチェックしているようです. MD5 [92] について衝突の脆弱性[93] が発見されたからでしょうか.

15.3 jman

システム標準の man コマンドは英語で記述されており, 読むのが大変なので, 日本語の jman をインストールしました.

```
cd /usr/ports/japanese/man/
make install
```

以上でインストール作業は終了です[94].

次に jman を使用するための設定を行いました. tcsh 向けに説明します. .xinitrc と似たようなものとして, .cshrc [95] があります. ここで環境変数[96] を設定しました.

```
% ee .cshrc
```

cshrc 内の似たような行の付近に次の記述を追加しました.

```
setenv LANG ja_JP.eucJP
setenv PAGER jless
```

.cshrc の再読み込みのため, 一旦ログアウトして再度ログインし jman jman を実行します. 正しく表示されれば成功です.

15.4 SCIM-Anthy

デスクトップ環境ですから当然日本語入力の必要があるため, 日本語入力のプログラムを入れました.

日本語入力まわりのプログラムの関係は, 歴史的な側面から非常に複雑です. 詳細な解説は他の書籍に譲るとして, 概要だけを説明します.

- ユーザは, 日本語入力を行う人間です. ex)mura
- X アプリは, 日本語入力を要求するエディタなどのプログラムです. ex)mlterm

[90] 本来ならこの姿勢はまずいですが, 原稿の締め切りがあるので.
[91] ハッシュ関数. さまざまなデータから擬似的に一意な値を吐き出す. 故意に同じ値を吐き出すようにすることは出来ない. というか出来てはまずい.
[92] これもハッシュ関数です.
[93] 故意に同じ値を吐き出すように出来てはまずいのに, その方法が発見されてしまいました.
[94] ports さまさまです.
[95] コンソールが表示されたとき, つまり, ログイン時に実行されます.
[96] プログラミングでの変数とほぼ同じです. プログラムは環境変数を読み込み, その動作を変更します.

- かな漢字変換サーバは, ひらがなから漢字への変換を専門に行います. ex)Anthy
- 日本語 FEP(IM とも呼ぶ) は, 上記 3 者の通信の仲介が専門です. ex)SCIM

複数の方法が乱立しており, どのプログラムがお勧めとは言えないのが現状です. 今回は独断と偏見と勘により SCIM と Anthy の組み合わせを選びました.

SCIM と Anthy を ports からインストールしました.

```
cd /usr/ports/japanese/scim-anthy/
# less pkg-message
--------------------------------------------------------------------------
Remember to set environment variables XMODIFIERS and LANG:

  csh/tcsh: setenv XMODIFIERS @im=SCIM  ; setenv LANG ja_JP.eucJP
  sh/bash:  export XMODIFIERS='@im=SCIM'; export LANG=ja_JP.eucJP

To start the SCIM input method daemon, use command:

  scim -d

--------------------------------------------------------------------------
# make install
```

インストール時にはパッケージの説明を読むことを忘れないようにしましょう. インストール後, 説明にしたがって環境変数を設定しました.

```
% cd
% ee .xinitrc
```

以下の内容を追加します.

```
export LANG=ja_JP.eucJP
export XMODIFIERS=@im=SCIM
export GTK_IM_MODULE=scim
echo '*inputMethod: SCIM' | xrdb -merge
scim &
```

ログインしなおして xinit すると日本語入力できるはずですが, 肝心の日本語入力対応のアプリケーションがまだ入っていないため試すことが出来ませんでした.

15.5 IPA フォント

SCIM-Anthy の make 中, CPU を遊ばせて置くのはもったいないと思い, IPA フォントをインストールしました. IPA フォントは IPA [97] が提供する高品質なフォントです.

[97] 独立行政法人 情報処理推進機構 http://www.ipa.go.jp/

```
cd /usr/ports/japanese/
ls | grep ipa
cd ipa-ttfonts
```

かなり時間がかかりましたが問題なく終了しました.

15.6 emacs

emacs をインストールしました. emacs に関しては特に説明することも無いでしょう.

```
cd /usr/ports/editor/emacs
make install
cd /usr/ports/editor/tamago
make install
```

emacs での日本語入力プラグインの tamago も make install しました.

xinit で X.Org を立ち上げた後, CtrlSpace を押し, emacs で日本語入力が出来ることを確認しました.

tamago での入力について調べたところ, 下記のような設定をすることで tamago なしで実現できることが分かりました.

```
% cat~/.emacs
; 日本語をデフォルトにする
(set-language-environment "Japanese")
; anthy.el をロードできるようにする
(push "/usr/local/share/emacs/site-lisp/anthy" load-path)
; anthy.el をロードする
(load-library "anthy")
; japanese-anthy をデフォルトの input-method にする
(setq default-input-method "japanese-anthy")
%
```

必要なくなった tamago はアンインストールしました.

```
cd /usr/ports/editor/tamago
make deinstall
```

15.7 sudo

emacs のインストール作業中, 平行して他の作業を行っていたときに, ターミナルに文章をペーストするという大ポカをやらかしてしまいました. 挙句の果てにそのターミナルが su した状態だったため, 背筋が凍るような[98]気がしました.

ターミナルが root でなければ大した脅威ではありません. そこで, sudo [99] を導入することにしました.

[98] root + 不明なコマンド = システムがどうなったか分からない

[99] % sudo make install のように使うと, そのコマンドだけ root として実行できる.

```
# cd /usr/sysutils/
# ls | grep sudo
# cd ..
# make search name="sudo" | less
# cd /usr/ports/security/sudo
# make install
```

問題なく make install が完了したので, 設定に移ります.

```
# cd /usr/local/etc
# ee sudoers
```

Same thing without a passwd の次の行のコメントを解除しました.

15.8 TEX

この文章も LATEX[100] で作成されています. その LATEX 環境を構築します.

```
% cd /usr/ports/japanese/teTeX
% sudo make install
```

先ほどの sudo コマンドを使用することで, 安全な一般ユーザのまま作業が出来ます. 特に問題なく make install が完了しました.

15.9 Texmaker

LATEX の統合開発環境[101] です.

```
% cd /usr/ports/editors/texmaker
% sudo make install
```

問題なく make install が完了しました. 日本語入力もまったく問題ありません.

この辺, 手抜きと思われる方もいらっしゃると思いますが, ports の出来が素晴らしすぎるため簡単にインストールができるため, 特に書くことがなくなってしまっているのです.

16 感想

ページ数から分かるとおり, ちょっと疲れました. もっと短くまとめられれば良かったのですが, デスクトップ環境の構築という大きなテーマを扱ったため, また, 私の能力不足もあり, この長さになりました. 長さの割に内容が薄いような気もちょっとします.

他に導入したアプリケーションも多々あり, その紹介が出来なかったのが残念です. 特に portupgrade [102] の紹介が出来なかったことが心残りです.

今回の記事を書き, 色々と勉強になりました.

[100] レポートから本まで何でも書ける素敵プログラム. 電通ではほぼ必修です.
[101] プログラミングでの Eclipse や Visual Studio みたいなもんです.
[102] ports の弱点であるアップグレード時の依存関係を面倒見てくれるソフト. コマンド一発ですべての ports を最新のものに更新できます.

第Ⅴ部
otenkiプロジェクトこの一年

(著者) oku

17　otenkiプロジェクトとは

otenkiプロジェクトは、現在電通大に設置されている、お天気情報システム[103]の相当品を作る企画です。
目標は、

- 各気象センサの出力をマイコンによって取得し、Ethernet経由で転送する。
- データベースに気象データを蓄積し、Web上での閲覧ができるようにする。

今年は検証のためにサークル棟屋外へのセンサ設置とコンピュータインターフェースの制作を行いました。

18　センサ設置

今回設置したセンサは、雨量計と風向風力計です。

18.1　雨量計

設置したセンサはCOT-34HT[104]です。
雨量計は倒れなければOKだろうということで、コンクリートブロックに穴を開けて、雨量計の足をボルトで固定した程度です。
問題は雨量計の足がプラスチック製で、この足で金属製の重い本体を支えることが出来るのだろうかと不安になったことと、まだ電源を配線していないので冬に凍結する虞れがあること位です。

18.2　風向・風力計

設置したセンサはCYG-5103[105]です。
付近にあったアマチュア無線で使っていたと考えられるアンテナ台に設置し、普通のアンテナと同じようにステイで固定しました。

19　コンピュータインターフェース

19.1　測定値の取得

19.1.1　風向・風力計

風向と風力はアナログな出力となっているので、PSoCをもちいてアナログ処理を行った後、シリアルで出力しています。
風向は風力計の水平方向の回転にたいして抵抗が変化するので、抵抗を測定して出力しています。
問題になったのは風力で、センサは正弦波として羽の回転を出力しますが、出力電圧の幅が広いので、アナログ回路部分の設計に苦労したようです。アンプで増幅したのち、閾値を横切るのを検出することでパルス出力に変換しています。

[103] http://weather.cc.uec.ac.jp
[104] http://www.weather.co.jp/PDF/COT-34T.pdf
[105] http://www.weather.co.jp/PDF/CYG-5103.pdf

19.1.2 雨量計

雨量計は内部に升を持っていて、これが振った雨で一杯になるとししおどしのように倒れて中の水を捨てるようになっています。実際には升が 2 つ並んで付いているので、2 つの升に交互に雨が溜まるようになります。

出力は、内部の升の転倒を単純にパルスで出力するので、マイコンでパルスを数えているだけです。

19.2 測定値の送信

さすがに風雪に晒されるところに PC を置くのは困るので、測定値はセンサ付近に設置するマイコンを使って、Ethernet 経由で PC に送信されます。

19.2.1 プロトコル

プロトコルには、いわゆる BSD-syslog を用いています。これは UDP で一定のフォーマットに従ったテキストデータを投げるだけなので、実装が容易で既存の syslog 実装を流用すれば[106]簡単にログが取れるのでらくちんです。

```
<131> SEQS:1181 3.4819135K ohm, 1.2837998 m/s.
<131> SEQH:2439 SHTH:01382 SHTT:06814 SHTE:0000
<131> SEQP:0813 PULA:0001 PULB:0001 PULC:6561 PULD:0001
```

実際には一行づつ送信します。すべてのメッセージはシーケンスを持ち、パケットの欠落や入れかわりに気づくことができるようになっています。SEQS が風向計の出力をデコードする PSoC の出力、SEQP の SEQA が雨量計のパルス出力になっています。

syslogd は次のように記録します。

```
Sep 27 00:09:57 <16.7> host SEQS:1181 3.4819135K ohm, 1.2837998 m/s.
Sep 27 00:10:01 <16.7> host SEQH:2439 SHTH:01382 SHTT:06814 SHTE:0000
Sep 27 00:10:01 <16.7> host SEQP:0813 PULA:0001 PULB:0001 PULC:6561 PULD:0001
```

19.2.2 H8 マイコンによるネットワーキング

今回の実装には秋月の H8/3069LAN ボードと TOPPERS/JSP と TINET を使いました[107]。

TOPPERS/JSP[108]は、uITRON4.0 スタンダードプロファイルに準拠した OS で、TINET[109]は ITRON TCP/IP 仕様 1.00.01 に準拠したネットワークプロトコルスタックです。

割り込みハンドラ中で Socket を触れないのに気づかなかった以外は特にハマるところもなく、echo のサンプルを多少修正するだけで簡単に実装できました。

が、ネットワークと関係ないところ (シリアルを受信していると唐突に止まる) で問題が起きていて未解決です。

20 To Do

- データベースに測定値を投げる
- H8 とカメラの屋上への設置

[106] それくらいの労力を惜しむなという声も有りそうだ。。
[107] 他に直接蟹を叩くコードも用意はしたけれど使わなかった
[108] http://www.toppers.jp
[109] http://www.toppers.jp/tinet.html