

2009年夏号

あんどきゅめんてつど
でびあん



東京エリアDebian勉強会
関西エリアDebian勉強会著

目次

1	Introduction	2
2	sqlite3 と python で csv ファイルを分析する	3
3	Debian JP 定例会議 on IAX	5
4	Git+ メールでの事前課題提出にまつわる課題	6
5	Namazu みたいに Google AJAX Search API	9
6	AspireOne で Debian sid サーバー	12
7	Debian GNU/Linux 2ch ブラウザ普及計画	14
8	Linux カーネルコンフィグ変換ツールを作ってみた	16
9	黒 MacBook の Lenny/Sid 64bit 化でのハマりポイント	21
10	Debian パッケージングハンズオンの手引書	23
11	Debian における Common Lisp プログラミング環境	33
12	Debian on chumby の作り方	39
13	Java ポリシーを読んでみた	46
14	advi をデバッグしてみた	50
15	ocaml に入門してみた	53
16	Erlang コードを Debian パッケージにしてみた	56
17	研究室のソフトウェアを Debian パッケージにしてみる	65
18	MC-MPI/GXP 公式パッケージへの道	76
19	Debian GNU/Linux 上で Android SDK を使ってみた	83
20	DDTSS 活用	89
21	パッケージの依存性を解く : EDOS から Mancoosi まで	91
22	Debian 勉強会の資料を作成しよう	95
23	LaTeX beamer でプレゼンテーション	99
24	各種イベント開催実績	103
25	2008 年を振り返ってみる	108
26	Debian Trivia Quiz	111
27	Debian Trivia Quiz 問題回答	114

1 Introduction

上川 純一

Debian 勉強会へようこそ。これから Debian の世界にあしを踏み入れるという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか？

Debian 勉強会の目的は下記です。

- Debian Developer (開発者) の育成。
- 日本語での「開発に関する情報」を整理してまとめ、アップデートする。
- 場 の提供。
 - 普段ばらばらな場所にいる人々が face-to-face で出会える場を提供する。
 - Debian のためになることを語る場を提供する。
 - Debian について語る場を提供する。

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりとするスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「場」としての空間を提供するのが目的です。



2 sqlite3 と python で csv ファイルを分析する

上川純一

sqlite はお手軽に SQL を利用するための仕組みです。データベースが一つの UNIX ファイルとして管理されており、データベースの作成・削除が簡便に行うことができること、また、サーバクライアントアーキテクチャではなく、OS の提供するファイルシステムのロック機構を活用して ACID 特性を実現しているという特徴があります (図 1)*¹。データベースを利用する場合においては、データベースをサーバクライアントモデルで利用しようとする、データベースファイル置き場やポート番号やホスト名やユーザ名やパスワードの設定が最低でも必要になりますが、それらが必要なくなります。そのため、データベースのインスタンスを別に立ち上げなくてもよいんだけど、ちょっと SQL を使いたいというようなクイックハックに便利です。



図 1 一般的な DB と sqlite の違い

まず、この記事に必要な関連パッケージをインストールしましょう。

```
$ sudo apt-get install sqlite3 python-pysqlite2
```

2.1 データベースの作成

sqlite3 という CUI のアプリケーションがあり、一般的な SQL 文を利用することができます。また、ruby, perl, ocaml, haskell, common lisp, Smalltalk などの一般的なプログラミング言語用のバインディングも用意されており、データベースを利用することができます。

まず、データベースを作成してみましょう。

```
$ sqlite3 debmtg.db
sqlite>
```

存在しない新しいファイル名を指定すれば、そのファイル名でデータベースが作成されます。この時点で SQL 文 (CREATE TABLE) などが利用できます。

*¹ <http://www.sqlite.org/atomiccommit.html> に仕組みの説明があります

2.2 データをつっこむ

データベースも作成できたので、データをつっこんでみましょう。csv ファイルからデータベースにデータを挿入するケースを考えてみます。実はsqlite3 の `.import` コマンドを使えばよいのですが、ここではプログラム言語のバインディングを活用してインポートしてみます。

まず、csv 形式でデータを用意します。

```
上川,10
岩松,15
山田,9
```

csv ファイルを読み込み SQL コマンドを出力する python のコードを書きます。

```
#!/usr/bin/python2.5

""" test program to import minimalistic csv file to sqlite database.
Creates debmtg.db
"""
from pysqlite2 import dbapi2
import csv

con = dbapi2.connect('debmtg.db')
cur = con.cursor()

cur.execute('create table test(name text, score number)')

for rows in csv.reader(file('test.csv')):
    cur.execute('insert into test(name, score) values(?,?)',
                (rows[0].decode('utf-8'),int(rows[1])))

con.commit()
con.close()
```

2.3 SQL を使ってみる

sqlite3 コマンドを実行するとインタラクティブに SQL 文を入力することが可能です。

まず、sqlite 独自の命令をつかってデータベースの構造を分析してみます。

```
$ sqlite3 debmtg.db
sqlite> .tables
test
sqlite> .dump test
BEGIN TRANSACTION;
CREATE TABLE test(name text, score number);
INSERT INTO "test" VALUES('上川',10);
INSERT INTO "test" VALUES('岩松',15);
INSERT INTO "test" VALUES('山田',9);
COMMIT;
```

SQL 文でランキングを調べたり平均値を調べたりもできます。

```
sqlite> select name, score from test order by score;
山田|9
上川|10
岩松|15
sqlite> select sum(score)/count(score) from test;
11
```

以上、簡単ですが、sqlite の紹介でした。

3 Debian JP 定例会議 on IAX

上川純一

Debian JP の定例会議は通常 IRC で行っているのですが、2009 年 1 月 8 日に執り行われた会議は実験的に音声通話も交えて行いました。利用したプロトコルは IAX プロトコルで、iaxcomm をクライアントとして利用し、asterisk サーバに全員接続しました。

MacBook のマイクまわりのハックが十分でなく、iaxcomm 自体の安定度合いもよくなかったので音質が悪いという問題がありましたが通信状態は概ね良好でした。今後さらに試験を重ねて実用的に使って行きたいものです。

4 Git+メールでの事前課題提出にまつわる課題

上川純一



2008 年 11 月、12 月、および 2009 年 1 月には事前課題を Git を利用して Git の生成するパッチをメールで提出してもらおうという形式にしていました。その場合には git pull するたびにコンフリクトが発生します。その原因となる仕組みと対策方法について説明します。

4.1 コンフリクトが発生しやすい理由

それでは、Debian 勉強会の 2008 年 11,12 月の資料でコンフリクトが発生しやすかった理由を分析してみましょう。

4.1.1 元のファイルの形式

```
-\subsection{}
+\subsection{XXXX}
+ 内容
+
```

同じファイルについて作業し、しかも同じ場所にすこしづつ異なる変更を別の人が行うという仕組みになっていました。この場合、二つのパッチはかならずコンフリクトし、手動で解消する必要があります。今回パッチが同じ場所に挿入している場合に、順序は問わないので適当でよいのですが、git apply にはその知識はないため、毎回コンフリクトの解消を行います。

4.1.2 メールを利用したサブミット

締切り直前に作業するのが習慣のため、複数の人がある時点の同じコミットに対してパッチを作成、それを上川がある時点でマージしました (図 2)。

毎回コンフリクトが発生するのですが、それを上川が解消してマージとして記録されています。そのため、各自が提出したバージョンとは若干違うパッチとなって alioth.debian.org にある Git ツリーにマージされています。

4.2 マージ・コンフリクトの発生のしかた

Git はデフォルトでは master ブランチで作業します。git pull コマンドは Alioth にあるリモートのツリーの情報を origin ブランチにとってきて、master ブランチにマージします。マージする内容がなければ、Fast-forward されます。

まず最初に各自が自分の Git ツリーの master ブランチでパッチを作成します。そして、git format-patch の結果をパッチファイルとして送付します。上川が git am でそのパッチを適用し、Alioth の Git ツリーに公開します。複数人が同時期に事前課題を提出しているため、git am でパッチを適用した場合にそのままでは適用できない状況がおき、「コンフリクト」が発生します。「コンフリクト」が発生した場合には上川が手動でその部分を修正します。

Graph	Short Log	Author	Author Date
●	finalize for printin...	Junichi ...	2008年12月19日 09時28分05秒
■	Merge branch '20...	Junichi ...	2008年12月19日 08時31分45秒
●	Add by Yamamoto	Hiroyuki...	2008年12月19日 07時05分21秒
■	Merge branch '20...	Junichi ...	2008年12月19日 08時29分55秒
●	キタハラ 事前課題	KITAHARA...	2008年12月19日 02時04分27秒
●	home work added...	Kazuyu...	2008年12月19日 06時21分06秒
■	use clearpage ins...	Junichi ...	2008年12月19日 00時41分40秒
●	tweak presentatio...	Junichi ...	2008年12月19日 00時05分40秒
■	Merge branch 'ma...	Junichi ...	2008年12月19日 00時01分43秒
●	Add my own answe...	Noritada...	2008年12月18日 23時24分46秒
●	initial presentation...	Junichi ...	2008年12月18日 23時59分04秒
●	add references to ...	Junichi ...	2008年12月18日 23時58分58秒
●	added by Aya Ko...	Aya Ko...	2008年12月18日 22時21分12秒
●	add comment by ...	Takuma...	2008年12月18日 08時54分42秒

図2 qgit でみたツリーの状況、手動でマージが複数発生している

各自が Alioth の Git ツリーの最新バージョンを `git pull` で取得してくると `origin` ブランチには上川の作成した変更を伴ったパッチが入ります。 `master` ブランチで自分の作成したものからは若干変更が加わっています。そのため、内容に整合性がとれるとはかぎらないため、コンフリクトが発生します。コンフリクトがおきなかったとしても履歴にマージが残ります。

4.3 マージ・コンフリクトの解消方法

4.3.1 rebase して履歴をきれいにする

マージなどを解消するためには、 `git rebase -i origin` で不要なパッチを目視して消す作業をすればよいです。そもそも `git pull --rebase` すると、おそらくマージが残らないが、コンフリクトの解消は必要になります。これは面倒です。

4.3.2 作業の仕方を変える

解決策としては、各自が提出用のブランチを作成してそこで作業し、そこで提出用のデータを作成してしまえばよいというのがあります。

```
$ git checkout -b preworkXXXX origin
$ # ... このブランチで作業
$ git format-patch ...

# 提出してしまったら master ブランチにもどる
$ git checkout master
$ git pull # 上川の適用した版がとりこまれる

# さらに新しいブランチを作成して次の月の作業を行う
$ git checkout -b preworkYYYY origin
```

preworkXXXX ブランチを毎回捨てる。

4.3.3 format-patch ではない方法を使う

もう一つの選択肢としては、 `format-patch` ではない方法でデータの送受信を行うことがあります。 `format-patch` と `git am` の送受信の過程でコミットのハッシュが異なっていることから各位のコンフリクトが発生しているのだろ

うということです。

git bundle でバイナリ形式でデータを送付することができます。これを使うとハッシュは保存されるので上川のコンフリクトは解消されませんが、コンフリクトを解消したログがそのまま残るので、各位のコンフリクトは発生しないでしょう。

4.4 リポジトリ管理者側の課題

視点をすこし変えてリポジトリの管理者として上川の作業を見てみましょう。git am でパッチを適用して、マージするという作業を行うのですが、そのワークフローは実は面倒です。

特にメールを受けて、git am がコンフリクトを起こす確率が高く、複数のパッチがあるとほぼ確実にコンフリクトを解消することになります。

現状は複数の一時的に利用するブランチに git am で適用したものをあとでマージする、という方法をとっています。

```
$ git checkout -b マージ用のブランチ A master
$ git am -3 パッチ
$ git checkout -b マージ用のブランチ B master
$ git am -3 パッチ
$ git checkout master
$ git merge マージ用のブランチ A マージ用のブランチ B
$ コンフリクトの解消
$ git commit -a
```

ただ、10人以上の事前課題のハンドリングには時間もかかり、毎回したい作業ではないです。

自動でコンフリクトするかしないかについては判定できるし、その結果がビルドするかはチェックできるのであれば、自動処理でコミットを管理できないか?と考えています。

最低限のチェックだけしてパッチを master ブランチに自動でとりこんでくれるような仕組み、ないものでしょうかねえ?

5 Namazu みたいに Google AJAX Search API

小室 文



自分のウェブサイトの検索用にわざわざ Namazu を導入しなくても Google の検索エンジンを使って手軽にサイト内全文検索する方法を説明します。

5.1 Google の検索エンジンをカスタマイズして使うには

利用するパターンがいくつかあります。それぞれのパターンについて設定方法を紹介します。

どれを選んでも基本は HTML ファイル(もしくは PHP など)をウェブサーバに置くことで動作します。JavaScript 版はブラウザさえあればどこでも動きます。

1. JavaScript あり
 - (a) 自分で JavaScript を組む
 - (b) ウィザードを使う
2. JavaScript なし

5.1.1 JavaScript あり・なし両方

Google AJAX Search API の KEY を事前に取得しておきます。

5.1.2 JavaScript あり

HTML ファイルの中で Google AJAX Search API JavaScript ライブラリをロードします。

```
<script src="http://www.google.com/uds/api?file=uds.js&v=1.0" type="text/javascript"></script>
```

```

//名前空間の為に追加
google.load("search", "1.0");

//検索をするオブジェクトを作成する
var searchControl = new google.search.SearchControl();

//検索準備
//子オブジェクトのサーチャーメソッドを検索コントロールに追加。必要であればオプションも。
var options = new GsearcherOptions(); //表示オプション指定
options.setExpandMode(GSearchControl.EXPAND_MODE_OPEN); //結果一覧を開いた状態で出力する
options.setExpandMode(GSearchControl.EXPAND_MODE_CLOSED); //結果を閉じた状態のまま結果出力をする
options.setExpandMode(GSearchControl.EXPAND_MODE_PARTIAL); //結果がオープン拡張モード
options.setRoot(document.getElementById("resultsComeHere")); //結果を指定する場所に表示するオプション

//サイト制限の設定
var siteSearch = new GwebSearch();
siteSearch.setUserDefinedLabel("Aya's Site");
siteSearch.setSiteRestriction("popowa.com"); //直接 URL を指定
siteSearch.setSiteRestriction("000455696194071821846:reviews"); //カ
スタムエンジンの Key を指定\footnote{カスタム検索エンジンとはウィザードで
検索窓のある程度のデザインや制限を付けて作ることが出来るものです。\\
\url{http://www.google.com/coop/cse/}}

//準備終了、セットする
searchControl.addSearcher(siteSearch, options); //オプションがあれば
searchControl.addSearcher(new GwebSearch()); //オプションがなければ
//出力準備
var result = document.getElementById("googleResults");
var drawOptions = new GdrawOptions(); //オプション用に drawOptions オブジェクトを作成
drawOptions.setDrawMode(GSearchControl.DRAW_MODE_LINEAR); //1) 一列に出力
drawOptions.setDrawMode(GSearchControl.DRAW_MODE_TABBED); //2) タブに出力
drawOptions.setSearchFormRoot(document.getElementById("searchForm")); //検索結果と検索フォームを切り離す機能
searchControl.draw(result, drawOptions);
//検索結果の保持 (prototype)
searchControl.setOnKeepCallback(this, MyKeepHandler); //この (this) 検索状況を MyKeepHandler に持たせておく。
searchControl.setSearchCompleteCallback(this, OnSearchComplete); //検索の実行後
searchControl.setSearchStartingCallback(this, OnSearchStarting); //検索の完了前
//検索する！
searchControl.execute(keyword);

```

そして、body 内に検索窓、検索結果を出力したい場所に div を入れます。

```

<div id="searchForm" class="searchForm">form</div>
<div id="googleResults" class="googleResults">googleResults</div>

```

5.1.3 JavaScript なし

http://ajax.googleapis.com/ajax/services/search/web に引数を渡してリクエストすると JSON 形式でレスポンスが得られます。

パラメータ	項目	注意事項
q?	検索したいキーワード、検索式	日本語だった場合は urlencode() しないとうまく動かない
v=1.0	プロトコル番号の指定	現在は 1.0 しかない
key?	Google AJAX Search API の Key	申請した FQDN=実際に使っている FQDN でなくてもよい
start?	検索結果の開始インデックス	ドメイン指定をすると 160 件から持って来てくれなくなる
cx?	カスタム検索エンジンの ID	特定のドメインからのみ検索する事が出来る
lr?	特定の言語のドキュメントを検索対象とする	

レスポンス形式

```

{"responseData": {
  "results": [
    {
      "GsearchResultClass": "GwebSearch",
      "unescapeUrl": "http://en.wikipedia.org/wiki/Paris_Hilton",
      "url": "http://en.wikipedia.org/wiki/Paris_Hilton",
      "visibleUrl": "en.wikipedia.org",
      "cacheUrl": "http://www.google.com/search?q\u003dcache:TwrPfh22hYJ:en.wikipedia.org",
      "title": "\u003cb\u003eParis Hilton\u003c/b\u003e - Wikipedia, the free encyclopedia",
      "titleNoFormatting": "Paris Hilton - Wikipedia, the free encyclopedia",
      "content": "\[1\] In 2006, she released her debut album..."
    },
    ...
  ],
  "cursor": {
    "pages": [
      { "start": "0", "label": "1" },
      { "start": "4", "label": "2" },
      { "start": "8", "label": "3" },
      { "start": "12", "label": "4" }
    ],
    "estimatedResultCount": "59600000",
    "currentPageIndex": 0,
    "moreResultsUrl": "http://www.google.com/search?oe\u003dutf8\u0026ie\u003dutf8..."
  }
}, "responseDetails": null, "responseStatus": 200}

```

レスポンスを個々のプログラムで処理する

```

<?php
function search($keyword, $page, $key){
    $uri = 'http://ajax.googleapis.com/ajax/services/search/web';
    $uri .= '?q=' . urlencode($keyword);
    $uri .= '&key='.$key;
    $uri .= '&v=1.0&rsz=large&hl=ja&start=' . $page;
    return json_decode(file_get_contents($uri));
}
$keyword = "東京エリア Debian 勉強会";
$key = 'アプリケーションのキー';
$data = search($keyword, 1, $key);
var_dump($data);
?>

```

5.1.4 現状の問題点

Google AJAX Search API のかえしてくる「検索結果」の総数には問題があります。

JavaScript あり: setSiteRestriction でドメインを指定して検索をしようとした場合、検索結果の総数が検索するたびに変わります。

JavaScript なし: ?q=DMC%\%20site:http://www.debian.or.jp/とするとドメイン検索は出来るが、start?で引数を渡すたびに同じように結果総数が変わります。

5.1.5 まとめ

Namazu みたいにページ分けされた検索の仕組みは構築出来ませんでした (2009/1 現時点)。

Google AJAX Search API の総数が変わる件については、estimatedResultCount というくらいなので、estimate なのでしょう。^{*2}ちなみに Yahoo Web 検索でも同じような問題が発生します。Yahoo!は検索リクエストがある度に検索結果を積算しているのだから違います、と予め免責してあります。^{*3}

^{*2} <http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=70920> 'How does Google calculate the number of results?' に記述があります

^{*3} 表示件数について: <http://help.yahoo.co.jp/help/jp/search/web/web-14.html>

6 AspireOne で Debian sid サーバー

id774

この冬休みを利用して AspireOne に Debian sid の環境を構築しました。AspireOne と言えば今流行りの NetBook としてモバイル用途で知られています。今回はその省電力性に注目し、モバイル以外にもポータブルで場所を取らない簡易サーバーとして利用可能なのではないかと考えトライしました。

6.1 AspireOne で sid を使ってみる

6.1.1 AspireOne の電力消費効率

AspireOne は省電力性に優れた Intel Atom CPU を搭載しており、アイドル状態で約 10W、高負荷状態でも 15W 程度で動作します。高性能を要するエンコードやリッピングではなく、ちょっとした Web サーバー等の用途であれば安価でエコなサーバーとして利用できそうです。

6.1.2 sid を選択する理由

通常、サーバーであれば安定版を選択します。しかし AspireOne ならいつでも携帯して持ち運び、管理者が対話的に操作してメンテナンスすることができるので、これなら最新のパッケージを利用できる sid を使っても、何かトラブルがあればすぐ対応できるでしょう。

6.2 Debian をインストール

Debian Wiki ^{*4} に情報がまとまっているので、基本的にこれを参考に作業しました。

6.2.1 名刺サイズのイメージからのインストール

sid のインストールには主に 2 種類の方法^{*5} があります。

- テスト版の Debian の sources.list を sid に書き換えて aptitude full-upgrade する。
- 最小サイズの最新ビルドイメージを利用してエキスパートモードでネットワークインストールをおこなう。

今回は後者の方法を採用しました。インストールには Daily ビルド最新版の Debian イメージ^{*6} を利用します。名刺サイズのインストールイメージを利用して、エキスパートモードでインストールを開始すると、不安定版を選択することができます。

^{*4} <http://wiki.debian.org/DebianAcerOne>

^{*5} <http://www.debian.org/CD/faq/#unstable-images>

^{*6} <http://cdimage.debian.org/cdimage/daily-builds/daily/arch-latest/i386/iso-cd/>

6.2.2 暗号化 LVM の設定

モバイルで外に持ち出すことを考えると、HDD や PC 本体の盗難によるリスクを考慮しなければなりません。Debian は etch から暗号化 LVM を標準で利用出来るようになりました。これで/boot 以外の領域を暗号化することができますのでセキュリティとしては非常に強力になるでしょう。

この際、元の Windows 領域を全て上書き末梢するので 4~5 時間程度かかります。そこで一晩放置して翌日作業をするようにしました。

6.2.3 無線 LAN の設定

AspireOne には Atheros AR5007 チップが搭載されています。そこで module-assistant を利用して MadWifi をインストールしました。

```
sudo aptitude install build-essential module-assistant madwifi-source
sudo m-a prepare
sudo m-a auto-install madwifi
```

6.2.4 イー・モバイルの設定

現時点で lenny/sid に搭載されているカーネル 2.6.26 ならイー・モバイルによる通信が可能で、さらに USB の着脱によるプラグアンドプレイにも対応しています。

pppconfig パッケージをインストールし /etc/ppp/peers/em ファイルに以下のように記述します。文字列はイー・モバイル固有ですのでそのまま指定します。dip グループに所属したアカウントなら pon em コマンドを発行すればイー・モバイルによるネットワーク接続ができるようになります。

```
user "em@em"
connect "/usr/sbin/chat -v -f /etc/chatscripts/pap -T *99***1#"
/dev/ttyUSB0
115200
noipdefault
usepeerdns
defaultroute
persist
noauth
```

6.3 sid をメンテナンスする

6.3.1 アップグレード

私は以下のコマンドの実行結果を cron で定期的を取得しています。

```
aptitude update && aptitude -s -v -y full-upgrade
```

実際にアップグレードはしませんが更新予定のパッケージの一覧を見ることができます。これに加え必要に応じて apt-listchanges や apt-listbugs 等を使い、既存パッケージの依存関係に影響を及ぼさないことを確認の上でアップグレードすることになっています。

6.3.2 仮想環境での事前検証

さらに確実性を高めるための方法として VMware など仮想環境を利用する方法があります。あらかじめゲスト OS として実機と同じパッケージ構成の sid を用意し、先にそちらを変更して問題が無いことを確認してから実機のパッケージを変更すればトラブルを避けることができます。

6.4 まとめ

Debian sid なら最新のドライバや機能が使えるので、流行の NetBook にインストールしてモバイルなサーバーとして活用するのも良いですね。ぜひみなさんもトライしてみてください。



7 Debian GNU/Linux 2ch ブラウザ普及計画

山本 浩之

結構前から色々な 2ch ブラウザを stable 用にバックポートして、2ちゃんねるスレッドテンプレなどで非公式に upload していました。しかし、最近 JD が公式パッケージ入りしましたが、それ以外はなかなか公式には入ってきません。そこでいくつかの 2ch ブラウザについて、公式パッケージ入りを検討してみました。

今回検討対象とした 2ch ブラウザは Navi2ch for Emacs、おちゅ～しゃ、Kita、Kita2、Chalice for Vim、w3m-2ch、Gnview です、それではそれぞれみてみましょう。

7.1 Navi2ch for Emacs

Emacs 上で動く 2ch ブラウザ。

ライセンス:GPL

昔から Debian Developer の野首さんが、Navi2ch の本家で最新の CVS 版のパッケージを配っていることで有名です。ライセンス的にも問題はありませんが、どうやら野首さんのポリシーとして、公式パッケージに入れるのは控えているそうです。現に、やっと Lenny 入りした JD も 2ch の仕様変更のため、Lenny 版は掲示板に書き込めない、ということになってしまいました。今後も続きそうな、いきなりの仕様変更にも対応できる配りかた (backports とか volatile とか) を検討すれば、きっと野首さんも公式パッケージに入れてくれるものと信じています。

7.2 おちゅ～しゃ

GTK+ 上で動く、C++ で書かれた 2ch ブラウザ。

ライセンス:2 項目 BSD (LGPL ライブラリを含む)

これはかなり前から検討していましたが、upstream の開発周期が不安定で、時々完全に止まってしまうのが難でした。最近、また開発が再開し、久々のアップデートが出ていたので、思わず ITP してしまいました。パッケージは CDBS を用いて、問題なくでき、起動も問題ないようです。man と README.Debian を書けばアップロード出来る所までできています。

7.3 Kita

KDE/QT 上で動く、C++ で書かれた 2ch ブラウザ。

ライセンス:GPL

これも前から非公式で配られていますが (<http://tossi.orz.hm>)、tossi さんにコンタクトを取った所、公式パッケージにするつもりはないそうです。私も検討しましたが、upstream が KDE4 への対応を、今のところする気が無いということなので、かなり速くて高機能なのですが、現在保留中です。

7.4 Kita2

KDE/QT 上で動く、Ruby で書かれた 2ch ブラウザ。

ライセンス:MIT/X

Kita の後継として出てきたブラウザです。まだまだバグもあり、機能も少ないのですが、ITP してみました。パッケージ的にも一応一通り揃え (CDBS)、スポンサー探しをしていたのですが、実は EUC-JP only な環境では警告もなく文字化けすることを上川さんに指摘され、upstream に連絡を入れている所です。

7.5 Chalice for Vim

Vim 上で動く 2ch ブラウザ。

ライセンス：俺俺ライセンス

開発も順調ですが、元々 Vim を Windows に移植している人が upstream なためか、以下のような'俺俺'ライセンスです。

利用許諾

以下の諸条件に同意された方へ本ソフトウェアの利用が許諾されます。

本ソフトウェアを利用することで、ソフトウェア利用者にはソフトウェア作成者へ対価を支払う義務が生じません。

本ソフトウェアの不具合がソフトウェア作成者へ報告された場合には、ソフトウェア作成者は期間を限定せずに本ソフトウェアを修正しますが、修正前後を問わず本ソフトウェアの利用に際して生じた損害をソフトウェア作成者は補償しません。

ソフトウェア利用者は本ソフトウェアを、商用・非商用を問わず、使用・再配布することができます。

ソフトウェア利用者へは本ソフトウェアを改変する権利がソフトウェア作成者より与えられます。但し本ソフトウェアへ改変を施したバージョンを再配布する場合には、改変内容及びその実装方法をソフトウェア作成者へ無条件で開示する義務が生じます。

以上の諸条件に同意できない場合は本ソフトウェアの利用を中止してください。

このうち、特に問題になるのは、「但し本ソフトウェアへ改変を施したバージョンを再配布する場合には、改変内容及びその実装方法をソフトウェア作成者へ無条件で開示する義務が生じます。」の部分だと思います。これを「ソフトウェア作成者」から、「公衆」に変えると多分 GPL が近くなるのではないかと考えていますが、その変更でありうる損害が良く分からず、まだライセンス変更交渉もしていません。

7.6 w3m-2ch

w3m 上で動く 2ch ブラウザ。

ライセンス：記述無し

半角カナ以外の書き込みはできることは確認しましたが、残念ながらライセンスも、upstream への連絡先も記述が無く、断念しました。

7.7 Gnview

GTK+ 上で動く perl で書かれた 2ch ブラウザ。

ライセンス:GPL

既にやまねさんにより ITP が出されていますが、日本語限定なこともあり、なかなかスポンサーが付かないようです。

7.7.1 現状の問題点

- 2ch ブラウザは日本語限定ソフトウェアで、日本語が分かるスポンサーを探さねばならない。
- 2ch は予告無く仕様変更をするため、新しいバージョンを即座に配布できる仕組みが必要。
- 2ch 自体の評判も芳しくなく、名前を出して関わることを嫌う人も多い。

などなど。

8 Linux カーネルコンフィグ変換ツールを作ってみた

岩松 信洋



8.1 はじめに

2008 年末に 2.6.28 が出ましたが、みなさんコンパイルしていますか？2、3 名ほどは毎朝昼晩 Linus ツリーから git pull してコンパイルされていると思いますが、大抵の方は Debian が提供しているカーネルを使っていると思います。使っている理由は様々ですが、コンフィグレーションがめんどくさいとか、どこを変えていいのかわからない、などが理由だと思います。^{*7}

また、最近で Linux カーネルも賢くなり、ドライバをモジュールにしておくと、ある程度自動的に必要なモジュールをロードしてくれるようになったのも理由の一つかもしれません。今回はユーザの立場からカーネルに触れるようになる方法の一つとして、Debian が提供しているカーネルからモジュールを組み込みするためのスクリプトを作りました。これによって、どこを有効にすればいいのかわからないなどの問題がすべて解決します。

8.2 なぜ彼らはカーネルをリコンパイルするのか？

普通の Debian ユーザはカーネルをリコンパイルしないようです。毎日狂ったようにリコンパイルしている人はカーネルハッカーか、変態さんぐらいでしょう。彼らがカーネルコンパイル！コンパイル！と言っているからには何か理由があると思います。カーネルをリコンパイルする理由として以下の事が考えられます。

- カーネルハックのため。
- カーネル BTS の深追い。
- 最新のカーネルは新しいドライバや機能が使えるから。
- ドライバを組み込みにして、起動の高速化。
- ドキドキ感を味わうことができる。
- 無駄に CPU を使いたい。(反エコ)

カーネルをリコンパイルすることはメリットだけではなく、デメリットもあります。

- 失敗したら動かなくなるかもしれない。
- コンパイルに CPU リソースを食いすぎる (CPU が遅いため)。

たぶん、この文章を読んでいる人たちは前者の予備軍なので、デメリットは気にせずに突き進んでいけると思います。

^{*7} i386 で約 4000 の設定項目がある

8.3 今回作ったプログラム

今回作ったプログラムは、システム情報を元にシステムに必要なカーネルモジュールが組み込み指定に変換されたカーネルコンフィグファイルを出力するというものです。

- 使うカーネル
Debian で提供しているカーネル (lenny では 2.6.26)
- 入力するデータ
 1. カーネルソースコード
 2. 動いているカーネルのコンフィグファイル
/boot/config-2.6.26-1-xxx
 3. システム情報
- 出力されるデータ
システム情報を元にシステムに必要なカーネルモジュールが組み込み状態になっているカーネルコンフィグファイル

8.3.1 システム情報の取得方法

今回のキモはどのようにして、システム情報を取得するか、にかかっています。今動いているカーネルから得られる情報は以下のものが考えられます。

コマンド	内容
dmidecode	BIOS からシステム情報を出力する
lspci	PCI の情報を出力する
lsusb	USB の情報を出力する
dmesg	カーネルデバッグメッセージを出力する
lsmod	ロードしてるモジュールを出力する

表 1 起動しているカーネルから得られる情報

これらの中で信用できて簡単に扱えるものは `lsmod` でしょう。理由はロードしているモジュール = 現在の Linux システムに必要なものなので、わかりやすいからです。よって、今回は `lsmod` の出力を利用することにしました。^{*8}

8.3.2 簡単な流れ

以下に簡単な処理の流れを説明します。

1. データとして、カーネルソースコードへのパス、動作しているカーネルコンフィグファイル、`lsmod` の出力結果を指定する。
2. `lsmod` からロードしているドライバモジュール一覧を取得する
`lsmod` を実行すると、以下のような内容が出力されます。

```
$ lsmod
Module                Size  Used by
i915                   25280  2
drm                    65256  3 i915
ipv6                   235300  10
rfcomm                 28272  2
l2cap                  17248  9 rfcomm
.....
```

^{*8} カーネルの自動認識がどこまで信用できるかはこの際無視します。

3. ドライバモジュール名 と modinfo コマンドから ドライバモジュールのパスを取得する。

modinfo コマンドを使うと、指定したドライバモジュール名の情報を取得することができます。-n オプションを使うと、ドライバオブジェクトファイルのパスが出力されます。

```
$ modinfo -n i915
/lib/modules/2.6.26-1-amd64/kernel/drivers/char/drm/i915.ko
```

上の結果を例にすると、/lib/modules/2.6.26-1-amd64/kernel/ 以下と カーネルソースコードのパス構造は同じなため、ドライバモジュール名 i915 の Makefile のあるパスは drivers/char/drm/Makefile になります。また、ドライバオブジェクトファイルは i915.ko であることが分かります。

4. 上で取得した Makefile へのパスとドライバオブジェクトファイル名より、対象になるドライバコンフィグ名を取得する。

ドライバオブジェクトファイル (i915.ko) とドライバモジュール名 (i915) は必ずしも一致するとは限らないので、ドライバモジュール名を使って、ドライバコンフィグ名を検索します。^{*9}

```
.....
obj-$(CONFIG_DRM_I830) += i830.o
obj-$(CONFIG_DRM_I915) += i915.o <- これ
obj-$(CONFIG_DRM_SIS) += sis.o
.....
```

5. 取得したドライバコンフィグ名を保存する。
6. 動作しているカーネルコンフィグファイルのドライバコンフィグを書き換える。

正規表現を使って書き換えると、以下のようになります。m はモジュールを意味し、y は組み込みを意味します。

変更前

```
.....
CONFIG_DRM_I830=m
CONFIG_DRM_I915=m
CONFIG_DRM_MGA=m
.....
```

変更後

```
.....
CONFIG_DRM_I830=m
CONFIG_DRM_I915=y <- 書き換え
CONFIG_DRM_MGA=m
.....
```

7. 変更したものをファイルに出力する。

8.3.3 実際に使ってみる

今回作成したソフトウェアは以下のように使います。ちなみに、lsmod の出力ファイルを指定しない場合は、プログラムの中で自動的に取得します。

```
$ moge -h
moge - Script to Kernel Module Enabler from lsmod command output

Copyright (C) 2008,2009 Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
Usage: moge [options]
  -c, --configfile <file>   Kernel config file name
  -k, --kernel               Kernel source path
  -o, --output <file>       outout file
  -l, --lsmod                lsmod command output file
  -h, --help                 display this help screen and exit
  -v, --version              show the version and exit

By Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
$ moge -o sage -c config-2.6.26-1-686 -l lsmod.list -k /usr/src/linux-2.6-2.6.26/
```

*10

^{*9} 例えば snd_hda_intel

^{*10} プログラム名が mogeなのはまだ決めていないためです。

8.4 作成されたコンフィグファイルを使ってカーネルをコンパイルする

さっそく、作成されたコンフィグファイルを使ってカーネルをコンパイルしてみましょう。カーネルコンパイルの方法は以下の通りです。

```
$ sudo apt-get update
$ sudo apt-get install linux-source-2.6.26 kernel-package
$ cd /usr/src/linux-source-2.6.26
$ make oldconfig
$ fakeroot make-kpkg --revision=yourpc00 kernel_image kernel_header
$ ls ../
linux-image-2.6.26_yourpc00_i386.deb
linux-headers-2.6.26_yourpc00_i386.deb
.....
```

8.5 変更結果

8.5.1 lsmod の結果

eeePC で試してみて、どれくらい変わったのか調べました。

変更前 61 モジュール

Module	Size	Used by
i915	25280	2
drm	65256	3 i915
ipv6	235300	10
rfcomm	28272	2
l2cap	17248	9 rfcomm
bluetooth	44900	4 rfcomm,l2cap
psmouse	32336	0
uvcvideo	45704	0
serio_raw	4740	0
compat_ioctl32	1312	1 uvcvideo
videodev	27520	1 uvcvideo
v4l1_compat	12260	2 uvcvideo,videodev
i2c_i801	7920	0
i2c_core	19828	1 i2c_i801
pcspkr	2432	0
itCO_wdt	9508	0
snd_hda_intel	324248	0
rng_core	3940	0
snd_pcm_oss	32800	0
snd_pcm	62596	2 snd_hda_intel,snd_pcm_oss
中略.....		
usb_storage	75936	0
sd_mod	22200	3
ata_piix	14180	2
ahci	23176	0
ata_generic	4676	0
libata	140384	3 ata_piix,ahci,ata_generic
scsi_mod	129356	3 usb_storage,sd_mod,libata
dock	8304	1 libata
ide_pci_generic	3908	0 [permanent]
ide_core	96168	1 ide_pci_generic
ehci_hcd	28428	0
uhci_hcd	18672	0
usbcore	118160	5 uvcvideo,usb_storage,ehci_hcd,uhci_hcd
thermal	15228	0
processor	32576	2 thermal
fan	4164	0
thermal_sys	10856	4 video,thermal,processor,fan

変更後 0 モジュール

Module	Size	Used by
--------	------	---------

8.5.2 bootchart

bootchart でどれくらい起動が速くなったか、調べてみました。

上の起動チャート図より、起動時のモジュールの読み込みがなくなり、三秒ほど速くなっていることが分かります。

8.6 今後の予定

このソフトウェアは Perl の勉強のために作っていましたが、今後の展開としては以下のことを考えています。

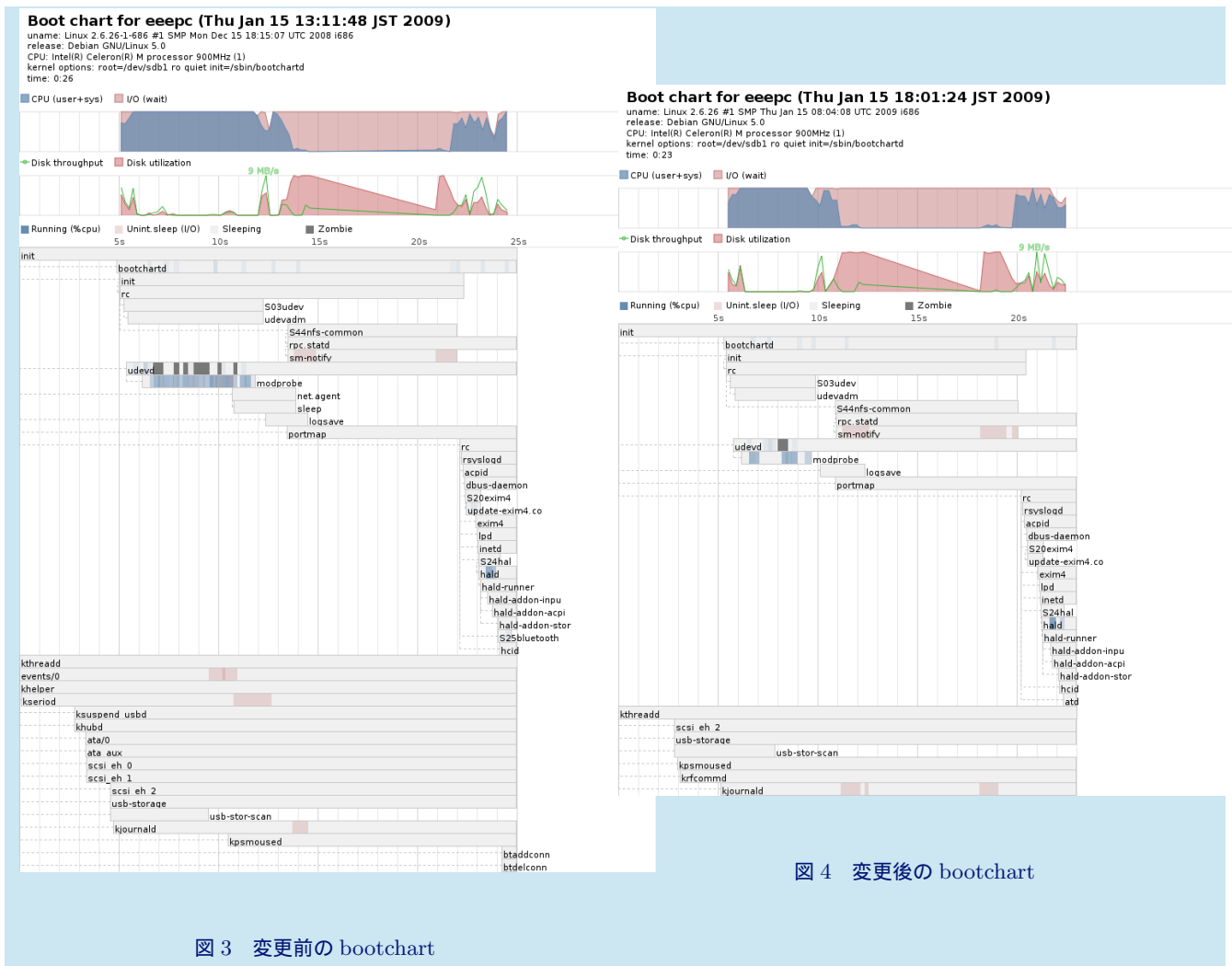


図 3 変更前の bootchart

図 4 変更後の bootchart

1. make-kpkg に入れる？
 make-kpkg は Perl で作られているので入れやすいかもしれません。
2. パッケージ化？
 Debian パッケージ化するとユーザは喜ぶかもしれません。
3. カーネルコンパイル Web サービスの提供？
 lsmod の出力が分かれば Debian カーネルはコンパイルが容易なので、サービス化するとユーザは喜ぶかもしれません。が、それぐらい自分でやれという感じです。
4. ドライバオブジェクトファイルとドライバモジュール名が一致しないやつを直す。
 知らないとハマるので、一致しておいた方がいいと思っています。

9 黒 MacBook の Lenny/Sid 64bit 化でのハマりポイント

まえだこうへい



年末のネタ*¹¹を実現するために、冬休みを利用して 32bit Sid 環境だった黒 MacBook を 64bit Sid で再構築しました。インストール自体はいつものとおりの作業なので、皆さんもいつもやっていらっしゃることで面白みもないのですが、年明けにリリースされていた Lenny の Snapshot イメージでブートローダに LILO を選択し、インストール後に起動させると Kernel Panic になってしまい、いきなり起動できない問題に遭遇します。今回はその回避方法について*¹²説明します。

9.1 用意したインストールイメージ

ご存知のとおり、MacBook を 64bit でインストールするには、amd64 版のインストールイメージが必要です。今回は、Lenny のスナップショット*¹³を使用しました。

9.2 簡単なインストール手順

MacBook への Debian のインストールに関する情報は、過去の Debian 勉強会の資料や、Debian Wiki に公開されています。今回もそれらと同様の手順です。

1. expert install での起動。
2. 日本語ロケール、米国キーボードを指定。
3. パーティションの指定し直し。*¹⁴
4. 基本パッケージのインストール。
5. シェルモードへ移行。
6. gptsync, refit パッケージのインストール、gptsync の実行。
7. apt-line の変更、apt-get {update,upgrade,dist-upgrade} の実行。
8. 再起動。

9.3 起動させると...

次のようなメッセージを吐いて、Kernel Panic になります。

*¹¹ 2008 年 12 月の勉強会で発表した、ヨメを Debian GNU/Linux Sid ユーザにする。

*¹² バグの修正ではありません。新婚旅行前日の数時間しか冬休みは時間を割けませんでした…。orz

*¹³ Debian GNU/Linux testing "Lenny" - Official Snapshot amd64 NETINST Binary-1 20090104-09:09

*¹⁴ LVM でボリュームを作っているパーティションはパーティションを切り直せない問題があります。

```
RAMDISK: Couldn't find valid RAM disk image starting at 0.
List of all partitions:
No filesystem could mount root, tried:
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(8,4)
```

実は、インストール直後に Kernel Panic となるのは今回が初めてではありません。MacBook Air を 64bit 化した際にも同じ現象が発生しました。^{*15}しかし、MacBook Air の時は、レスキューモードで起動し、Kernel を再構築してインストールしたら再発しなくなりました。ところが今回は Kernel を再構築しただけでは解決できません。困ったものです。

9.4 回避方法

困ったので、ググってみたところ、同じ現象に対する回避策も公開されていました。^{*16}簡単に回避策をまとめると次の手順です。

1. レスキューモードで起動します。
2. シェルモードになり、/target へ chroot します。
3. カーネルソースを展開します。
4. /boot 以下の該当の kernel config をカーネルソースツリーにコピーします。
5. kernel を構築し、インストールします。

```
REVISION=$(date +%Y%m%d.%H%M)
make-kpkg --initrd --revision $REVISION kernel_image
dpkg -i ../kernel-image-2.6.26_20090105.2330_amd64.deb
```

6. /ramdisk ディレクトリを作り^{*17}、initrd を展開します。

```
mkdir /ramdisk
cd /ramdisk
zcat /boot/initrd-2.6.26 | cpio -i
```

7. 先ほどインストールした kernel をアンインストールします。

```
dpkg --purge kernel-image-2.6.26
```

8. .config の "INITRAMFS_SOURCE" を書き換え、-initrd オプションなしで kernel をリビルドします。

```
sed -i 's:INITRAMFS_SOURCE="":INITRAMFS_SOURCE="/ramdisk":' .config
make-kpkg --revision $REVISION kernel_image
```

9. できた kernel パッケージをインストールします。
10. lilo.conf の "initrd=/initrd.img" をコメントアウトし、lilo を書き込みます。

これで、kernel panic を起こさず、ちゃんと起動できるようになりました。ただし、今のところ、新たなバージョンの kernel を構築する度、同じ手順を踏まなければなりません。

9.5 もっと簡単な回避方法。

lilo なんかをやめて、grub2 にしてしましましょう。おあとがよろしいようで。

^{*15} <http://d.hatena.ne.jp/mkouhei/20080713/1215913997>

^{*16} <http://linux.derkeiler.com/Mailing-Lists/Debian/2008-11/msg01918.html>

^{*17} ディレクトリ名は任意。/ 直下に所有者、グループを root:root で作ります。

10 Debian パッケージングハンズオン の手引書

岩松 信洋



10.1 本日の目的

Debian パッケージ化されていないソフトウェアをパッケージ化して、ビルドテストとパッケージの変更までを体験します。ところどころにトラップがあるので注意しましょう。

10.2 本日の流れ

1. 講師紹介
2. 作業を始める前の前準備
3. ソフトウェアのコンパイル
4. パッケージの雛形
5. CDBS
6. debian ディレクトリ以下ファイルの編集
7. パッケージのビルド
8. パッケージのインストール
9. パッケージのビルドテスト
10. パッケージのインストール/アンインストールテスト
11. プログラムの編集
12. 質疑応答

10.3 記号の説明

\$ が付いている場合は、コンソールからの入力を意味します。\$は入力せずにコマンドを入力してください。

コマンドラインやファイルの中身で \ が書かれている場所は行が続いている事を意味します。入力しないでください。

... は省略を意味します。実際には長い出力がある場合に省略している場合に利用しています。

10.4 エディタ

本ハンズオンでは、エディタとして vi および mousepad を使えるようにしています。vi が使えない人は、mousepad を使ってください。Windows のメモ帳と同じ機能を持ったエディタです。

10.5 ルート権限について

本ハンズオンでは、root 権限を使った作業を行う場合があります。その場合には sudo コマンドを使って作業をします。sudo コマンドが必要な場合にはコマンドラインの説明のところに sudo を指定しています。

10.6 前準備

10.6.1 パッケージメンテナ名の設定

パッケージメンテナの名前とメールアドレスを環境変数に設定します。適当なエディタを使って、`/home/user/.bashrc` に以下の例のように変更して保存してください。各項目には自分の名前とメールアドレスをいれてください。

```
export DEBFULLNAME="Nobuhiro Iwamatsu"
export DEBEMAIL=iwamatsu@nigauri.org
```

保存できたら、ターミナルを起動し、

```
$ source ~/.bashrc
```

を実行してください。

10.6.2 web サーバの立ち上げ

コンソールから以下のコマンドを実行してください。これは Live-CD 環境で apt-get ができるようにするための対策として行っています。実際のパッケージ作成では必要ありません。

```
$ sudo ruby1.8 ./tools/web.rb
```

10.6.3 apt-line の変更

エディタを使い、`/etc/apt/sources.list` ファイルを以下のように変更してください。apt-line が書かれていますが、削除してください。

```
deb http://localhost/debian lenny main
```

10.6.4 リポジトリ情報のアップデート

リポジトリのアップデートを行います。以下のようにコマンドを実行します。

```
$ sudo apt-get update
```

10.6.5 /tmp のマウントオプションの変更

`/tmp` を `nodev` オプションなしで `remount` します。以下のように実行します。

```
sudo mount -o remount,dev /tmp
```

10.7 今回のサンプル

今回は、`cwidget` を使ったサンプルプログラム `/live/image/osc/data/hello-cwidget-0.1.tar.gz` を用意しました。このサンプルプログラムを Debian パッケージ化します。`/live/image/osc/data` ディレクトリにソースファイルがあるので、ホームディレクトリに展開します。

```
$ cd
$ tar -xzf /live/image/osc/data/hello-cwidget-0.1.tar.gz
```

*18

このソフトウェアは C++ で記述されており、コンパイルに必要なライブラリやソフトウェアがインストールされている場合には、`./configure ; make ; make install` でコンパイルおよびインストールまでができるようになっています。

10.8 パッケージ化開始

10.8.1 ソースを読んでみる

動作しないプログラムをパッケージ化してもしょうがないので、先にどのようなソフトウェアなのか理解するためにもパッケージ化する前にソースコードを読んで、ソフトウェアの中身を理解して置きましょう。

10.8.2 とりあえず、コンパイルしてみる

動かないプログラムをパッケージ化してもしょうがないので、動作確認をします。まずは最低限コンパイルに必要なパッケージをインストールする必要があります。それが `build-essential` パッケージです。これは、パッケージ化の場合にも必要です。以下のように実行し、インストールします。

```
$ sudo apt-get install build-essential
```

先ほど解凍したディレクトリに移動します。移動したら、`configure` を実行します。

```
$ cd hello-cwidget-0.1
$ ./configure
...
Alternatively, you may set the environment variables \
SIGC_CFLAGS
and SIGC_LIBS to avoid the need to call pkg-config.
See the pkg-config man page for more details.
...
```

実行すると、エラーになります。

10.9 必要なライブラリを探す

Debian で特定のファイルが提供されているパッケージを探す場合には、`apt-file` を利用します。以下のように実行し、インストールします。

```
$ sudo apt-get install apt-file
```

通常は、この後、`apt-file update` を実行し、ファイル情報データを取得しますが、既に Live-CD に入れているので省略します。ファイルを探すには以下のように実行します。

```
$ apt-file search pkg-config
...
nant: /usr/share/doc/nant/help/functions/pkg-config.\
    is-max-version.html
pkg-config: /usr/bin/pkg-config
pkg-config: /usr/share/doc/pkg-config/AUTHORS
...
```

実行すると、指定したファイルを提供しているパッケージ名が出力されます。出力されたパッケージをインストールします。

```
$ sudo apt-get install pkg-config
```

再度 `configure` を実行してみましょう。

*18 このファイルは <http://www.nigauri.org/~iwamatsu/trash/hello-cwidget-0.1.tar.gz> からダウンロード可能です。

```
$ ./configure
...
No package 'sigc++-2.0' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.
...
```

まだ足りないパッケージがあるようです。先ほどと同じように apt-file を利用して検索し、インストールします。

```
$ apt-file search sigc++-2.0.pc
libsigc++-2.0-dev: /usr/lib/pkgconfig/sigc++-2.0.pc
$ sudo apt-get install libsigc++-2.0-dev
```

再度 configure を実行します。

```
$ ./configure
...
checking for CWIDGET... configure: error: Package \
  requirements(cwidget) were not met:

No package 'cwidget' found

Consider adjusting the PKG_CONFIG_PATH environment variable \
if you
installed software in a non-standard prefix.
...
```

エラーになります。まだ足りないようなので、再度検索してインストールします。

```
$ apt-file search cwidget.pc
libcwidget-dev: /usr/lib/pkgconfig/cwidget.pc
$ sudo apt-get install libcwidget-dev
```

```
./configure
...
config.status: WARNING:  Makefile.in seems to ignore the \
  --datarootdir setting
config.status: creating src/Makefile
config.status: WARNING:  src/Makefile.in seems to ignore the \
  --datarootdir setting
config.status: creating config.h
```

configure が正常に終了しました。終了すると、Makefile が作成されています。make を実行し、コンパイルします。

```
$ make
...
make[1]: ディレクトリ '/home/user/hello-cwidget-0.1' \
 に入ります
Making all in src
make[2]: ディレクトリ '/home/user/hello-cwidget-0.1/src' \
 に入ります
g++ -DHAVE_CONFIG_H -I. -I. -I.. -g -O2 -I/usr/ \
  include/sigc++-2.0 \
  -I/usr/lib/sigc++-2.0/include -I/usr/lib/cwidget \
  -I/usr/include/sigc++-2.0 -I/usr/lib/sigc++-2.0/include \
  -c hello.cc
g++ -g -O2 -I/usr/include/sigc++-2.0 -I/usr/lib/ \
  sigc++-2.0/include \
  -I/usr/lib/cwidget -I/usr/include/sigc++-2.0 \
  -I/usr/lib/sigc++-2.0/include \
  -o hello hello.o -lsigc-2.0 -lcwidget -lncursesw \
  -lsigc-2.0
make[2]: ディレクトリ '/home/user/hello-cwidget-0.1/src' \
  から出ます
make[2]: ディレクトリ '/home/user/hello-cwidget-0.1' に入ります
make[2]: ディレクトリ '/home/user/hello-cwidget-0.1' から出ます
make[1]: ディレクトリ '/home/user/hello-cwidget-0.1' から出ます
```

コンパイルも正常に終了したので、試しに実行してみます。

```
$ ./src/hello
```

ここまではサンプルプログラムの動作確認です。動作しないプログラムをパッケージ化してもしようがないので、先にどのようなソフトウェアなのか理解するためにもパッケージング化する前にソースコード等を読んでおくことをお勧めします。

10.10 Deban パッケージの雛形

`dh_make` コマンドでパッケージの雛形を作成することができます。`dh_make` は、`dh-make` パッケージで提供されています。以下のコマンドを実行し、インストールします。

```
$ sudo apt-get install dh-make
```

雛形の作成は以下のコマンドを実行します。

```
$ dh_make --createorig -s
```

`-createorig` オプションはオリジナルソースコードの `tar.gz` イメージを構築します。今回はシングルバイナリパッケージ（一つのソースコードから一つのバイナリパッケージが作成される）なので `-s` を指定します。実行すると以下のようなメッセージが表示されるので、Enter キーを押します。

```
Maintainer name : Nobuhiro Iwamatsu
Email-Address   : iwamatsu@nigauri.org
Date            : Sun, 15 Feb 2009 23:51:58 +0900
Package Name    : hello-cwidget
Version         : 0.1
License        : blank
Using dpatch    : no
Using quilt     : no
Type of Package : Single
Hit <enter> to confirm:
```

10.10.1 debian ディレクトリ

うまく動作すると、`debian` ディレクトリが作成され、この中に雛形が作成されます。パッケージメンテナはこのディレクトリの中以外は触りません。以下のような状態になっています。

```
.
|-- README.Debian (Debian パッケージの README)
|-- changelog     (Debian パッケージのチェンジログ)
|-- compat       (Debian パッケージのバージョン)
|-- control      (Debian パッケージ情報)
|-- copyright    (コピーライト情報)
|-- cron.d.ex    (cron を使うパッケージ用設定ファイル)
|-- dirs         (作成するディレクトリ名を指定する)
|-- docs         (インストールするドキュメントファイルを指定する)
|-- emacsen-install.ex (emacs 用設定ファイル)
|-- emacsen-remove.ex (emacs 用設定ファイル)
|-- emacsen-startup.ex (emacs 用設定ファイル)
|-- hello-cwidget.default.ex (debfont 用)
|-- hello-cwidget.doc-base.EX (doc-base 用)
|-- init.d.ex    (init.d を使うパッケージ用設定ファイル)
|-- init.d.lsb.ex (init.d を使うパッケージ用設定ファイル)
|-- manpage.1.ex (manpage の雛形)
|-- manpage.sgml.ex (manpage の雛形)
|-- manpage.xml.ex (manpage の雛形)
|-- menu.ex     (メニューの雛形)
|-- postinst.ex (postinst メンテナファイルの雛形)
|-- postrm.ex  (postrm メンテナファイルの雛形)
|-- preinst.ex (preinst メンテナファイルの雛形)
|-- prerm.ex   (prerm メンテナファイルの雛形)
|-- rules     (パッケージビルドスクリプト)
|-- watch.ex  (アップストリームチェック用ファイル)
```

10.11 CDBS

`./configure ; make ; make install` でパッケージのコンパイルができるソフトウェアは `cdbs` を使った方が容易に Debian パッケージ化できます。

10.11.1 一回 hello-cwidget ディレクトリを削除する

現状では先ほどの `dh_make` の結果が残っているので一回、サンプルプログラムのディレクトリごと削除し、再度展開します。

```
$ cd
$ rm -rf hello-cwidget-0.1.*
$ tar -xzf /live/image/osc/data/hello-cwidget-0.1.tar.gz
$ cd hello-cwidget-0.1
```

10.11.2 dh_make を実行し、パッケージの雛形を作成する

CDBS を使う Debian パッケージの雛形作成は以下のコマンドを実行します。

```
$ dh_make --createorig -b
```

-b オプションを指定すると、CDBS を使った雛形を作成します。以下のようなメッセージが表示されるので、エンターキーを押します。

```
Maintainer name : Nobuhiro Iwamatsu
Email-Address   : iwamatsu@nigauri.org
Date            : Sun, 15 Feb 2009 23:51:58 +0900
Package Name    : hello-cwidget
Version         : 0.1
License         : blank
Using dpatch    : no
Using quilt     : no
Type of Package : cdb
Hit <enter> to confirm:
```

10.11.3 不要なファイルの削除

今回のパッケージ化に必要なではないファイルを debian ディレクトリ以下から削除します。

```
$ rm -rf debian/*.ex debian/*.EX
```

10.11.4 debian/changelog ファイルの編集

debian/changelog ファイルには ITP(Intent To Package) のバグが既にかかれていて削除します。以下のように変更します。

```
hello-cwidget (0.1-1) unstable; urgency=low

* Initial release.

-- Nobuhiro Iwamatsu <iwamatsu@nigauri.org> \
   Wed, 18 Feb 2009 16:31:25 +0000
```

10.11.5 debian/copyright ファイルの編集

```
This package was debianized by Nobuhiro Iwamatsu \
  <iwamatsu@nigauri.org> on
Wed, 18 Feb 2009 16:31:25 +0000.

It was downloaded from <http://www.nigauri.org/~iwamatsu/>

Upstream Author:

  Nobuhiro Iwamatsu <iwamatsu@nigauri.org>

Copyright:

  Copyright (C) 2009 Nobuhiro Iwamatsu <iwamatsu@nigauri.org>

License:

  GPLv2

The Debian packaging is (C) 2009, Nobuhiro Iwamatsu \
  <iwamatsu@nigauri.org> and
is licensed under the GPL, see '/usr/share/common-licenses/GPL'.
```

10.11.6 debian/control ファイルの編集

```
Source: hello-cwidget
Section: devel
Priority: extra
Maintainer: Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
Build-Depends: cdb, debhelper (>= 7), autotools-dev
Standards-Version: 3.8.0
Homepage: http://www.nigauri.org/~iwamatsu/

Package: hello-cwidget
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Debian Packaging Hands-on sample program
This is sample program of Debian Hands-on done with
OSC2009 TOKYO Spring.
This is very easy program that uses CWidget.
```

10.11.7 パッケージのビルド

パッケージのビルドには `debuild` コマンドを使います。 `debuild` コマンドは `devscripts` パッケージで提供されています。また、まだ `CDBS` パッケージをインストールしていないので、一緒にインストールします。パッケージをインストールしたら、パッケージのビルドを試みましょう。

```
$ sudo apt-get install devscripts cdb
$ debuild -us -uc
...
dpkg-buildpackage: full upload (original source is included)
Now running lintian...
W: hello-cwidget: binary-without-manpage usr/bin/hello
W: hello-cwidget: new-package-should-close-its-bug
Finished running lintian.
```

10.12 パッケージのインストール

パッケージが無事ビルドできたら、実際にインストールしてみます。インストールには `dpkg` コマンドを使ってインストールします。インストールしたら、実際に動くか確認してみましょう。

```
$ sudo dpkg -i ../hello-cwidget_0.1-1_i386.deb
$ which hello
$ hello
```

10.13 パッケージのビルドテスト

パッケージができたあとはパッケージのテストを行います。パッケージのビルドテストには `pbuilder` を使います。 `pbuilder` は Debian に必要な最低限の環境からビルドを行い、依存関係等のチェックを行ってビルドテストを行うツールです。

10.13.1 pbuilder パッケージのインストール

```
$ sudo apt-get install pbuilder
```

10.13.2 pbuilder 環境の構築

ビルドテストを行う前に `base` システムイメージを構築する必要があります。通常は以下のように実行しますが、

```
$ sudo pbuilder --create --distribution lenny
```

今回はメモリの制限があるため、既に用意してある `base` システムイメージを利用します。イメージは `/live/image/osc/data/base.tgz` にあります。

10.13.3 パッケージのビルドテスト

pbuilder でテストする場合には作成されたパッケージの dsc ファイルを指定します。このファイルには、Debian パッケージの構成に必要なファイル名が書かれているので、その情報を元に再ビルドを行うことができます。また、実行前に apt-get clean コマンドを実行してキャッシュをクリアしてください。メモリが足りないためです。

```
$ cd ..
$ sudo apt-get clean
$ sudo pbuilder --build --distribution lenny \
  --basetgz /live/image/osc/data/base.tgz \
  --buildplace /tmp hello-cwidget_0.1-1.dsc
...
```

10.13.4 なぜエラーになるのか

先ほどの手順でやってもビルドエラーになります。なぜエラーになるのでしょうか。考えてみましょう。

10.13.5 再ビルドテスト

エラーになる理由は先にインストールしたパッケージ libcwidge-dev をパッケージビルド時の依存関係を記述するフィールド Build-Depends に追加していないためです。追加して、再ビルドしてみます。再ビルドには以下のように実行します。今度はうまくビルドができるはずですよ。

```
$ sudo pdebuild -- --distribution lenny --basetgz \
  /live/image/osc/data/base.tgz --buildplace /tmp
```

10.14 パッケージのインストール/アンインストールテスト

パッケージがビルドできただけでは喜んではいけません。インストール/アンインストールのテストも行いましょう。パッケージのインストール/アンインストールのテストには piuparts パッケージを使います。

10.14.1 piuparts のインストール

以下のように実行し、インストールします。

```
$ sudo apt-get install piuparts
```

10.14.2 パッケージのインストール/アンインストールテスト

piuparts も pbuilder と同様に最低限の環境からのインストールをチェックします。よって、base システムイメージが必要です。普段は指定する必要はありませんが、今回は -b オプションを付けて、/live/image/osc/data/base.tgz にある base システムイメージを指定して実行します。

```
$ cd ..
$ sudo piuparts -d lenny -b /live/image/osc/data/base.tgz \
  hello-cwidget_0.1-1_i386.deb
...
0m41.9s DEBUG: Removed directory tree at /tmp/tmpHliOKO
0m41.9s INFO: PASS: All tests.
0m41.9s INFO: piuparts run ends.
```

10.15 プログラムの編集

hello-cwidget を実行して、違和感のある方がおられたと思います。そう、Lenny がリリースされたというのに Etch になっていました。これはよくないので変更してみます。今回はよく利用されている dpatch を使って説明します。

10.15.1 dpatch のインストール

dpatch をインストールするには、以下のように実行します。

```
$ sudo apt-get install dpatch
```

10.15.2 dpatch を使うための準備

dpatch を使う前に、`debian/rules` ファイルに dpatch を使うように設定する必要があります。dpatch は一回、パッケージの状態を初期化してから行うためです。`hello-cwidget-0.1` ディレクトリに移動して、`debian/rules` を以下のように修正します。

```
$ cd hello-cwidget-0.1
```

```
#!/usr/bin/make -f
include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/autotools.mk
include /usr/share/cdb/1/rules/dpatch.mk
include /usr/share/dpatch/dpatch.make
```

10.15.3 dpatch の実行

dpatch は自パッケージを一回コピーし、dpatch 環境に移行します。その中で変更して、dpatch 環境を終了する時に差分を作成します。dpatch 環境に移行するには `dpatch-edit-patch` コマンドに作成する差分を保存するファイル名を指定して実行します。以下のように実行してください。

```
$ dpatch-edit-patch 01_change_dist
```

10.16 ファイルの変更

今回変更するファイルは `src/hello.cc` です。エディタを起動し、対象のファイルを変更します。mousepad の場合は以下のように実行します。

```
$ mousepad ./src/hello.cc
```

Etch の部分を Lenny に変更したあと、保存してエディタを終了します。

10.16.1 dpatch 環境を終了する

dpatch 環境を終了するには以下のように実行してください。実行すると、差分をファイルに保存して dpatch 環境を終了します。

```
$ exit
```

10.16.2 作成された差分 (patch) の中身

作成された差分は

`debian/patches/01_change_dist.dpatch` として保存されています。以下のような内容になっているはずです。

```

#!/bin/sh /usr/share/dpatch/dpatch-run
## 01_change_dist.dpatch by Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
##
## All lines beginning with '## DP:' are a description of the patch.
## DP: No description.

@DPATCH@
diff -urNad hello-cwidget-0.1~/src/hello.cc hello-cwidget-0.1/src/hello.cc
--- hello-cwidget-0.1~/src/hello.cc 2009-02-15 06:56:01.000000000 +0000
+++ hello-cwidget-0.1/src/hello.cc 2009-02-18 16:54:40.668274925 +0000
@@ -26,7 +26,7 @@
    toplevel::init();

    widgets::widget_ref dialog =
-    dialogs::ok(L"Hello, Debian GNU/Linux Etch!",
+    dialogs::ok(L"Hello, Debian GNU/Linux Lenny!",
        util::arg(sigc::ptr_fun(toplevel::exitmain)));

    toplevel::settoplevel(dialog);

```

パッチにはなぜそのような説明をしたのか、説明を書く必要があります。## DP: No description. の部分に説明を書きます。以下のように変更するといいかもかもしれません。

```
## DP: Change distributin name from Etch to Lenny.
```

10.16.3 作成した差分をパッケージに反映させる

差分は作成されましたが、このままではパッケージ作成時に差分が適用されません。dpatch を使って差分をパッケージに適用させるには debian/patches/00list ファイルを作成し、パッケージにパッチをファイルに列挙する必要があります。debian/patches/00list を以下のように変更します。

```
01_change_dist.dpatch
```

10.16.4 差分を適用したパッケージを作成する

差分を適用したパッケージを作成するには通常のパッケージ作成と変わりません。debuild コマンドを使って作成します。

```
$ debuild -us -uc
....
```

10.16.5 パッケージ作成エラーになる

説明どおりに操作している人は、パッケージ作成エラーになると思います。理由は何なのか、考えてみましょう。原因が分かった人は、再ビルドした後に、実際にインストールして、差分が反映されているか確認してください。もちろん pbuilder/piuparts を使ってパッケージのテストを行う事も忘れずに。



11 Debian における Common Lisp プログラミング環境

日比野 啓

11.1 Common Lisp はどんな言語か

Debian が Common Lisp のために用意している仕組みが何を目的としているのか、なぜそのような仕組みが用意されているのかを理解するために、まずは Common Lisp がどんな言語でライブラリをどのように構築しているのかを説明します。

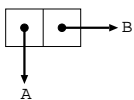
11.1.1 Lisp のプログラムの構文

Lisp のプログラムは、S 式と呼ばれる、トークンと入れ子の括弧の列で表現されます。表記と構造を対応づけて説明するために、まずはドット . を使った表記から説明します。

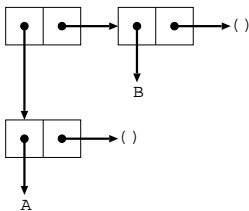
```
S-exp : () | token | (S-exp . S-exp)
```

S 式は 空の括弧 () かトークンか S 式のペア (ドットをはさんで括弧でくくったもの) ということになります。S 式のペアの構造を考えるには以下のようなポイントの組を考えると構造的な理解がしやすいです。

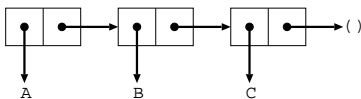
```
(A . B)
```



```
((A . ()) . (B . ()))
```



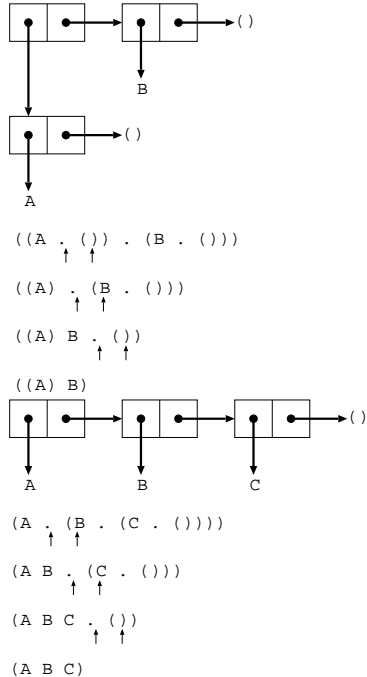
```
(A . (B . (C . ())))
```



このポイントのペアのことを Lisp では cons セルと呼びます。左側のポイントは car、右側のポイントは cdr と呼

びます。

cdr が cons セルあるいは空の括弧 () を指している場合は . と cdr 内の括弧 () を省略できます



このように cdr で連なる連結リストを簡潔に表現することができます。空の括弧 () は要素が一つもない空のリストということです。Common Lisp では空リスト () は nil と書くこともできます。この . の省略まで含んだのが一般的に S 式と呼ばれている表記です。

Lisp の処理系はこのリスト構造で表現されるプログラムを処理することで実現できます。LISP は LIST Processing language の略だというわけです。LISP のプログラムがリスト構造と等価であるということが今回の話の重要なポイントなので注意しておいてください。

11.1.2 Common Lisp のプログラム

では実際に Lisp のプログラムを見ていきましょう。ここから先は実際に対話環境でプログラムを試しながら説明していきます。CL-USER>というのが対話環境のプロンプトです。

関数の呼び出し ではもともと定義されている関数を呼び出してみます。足し算を行なう関数 + の例です。

```
CL-USER> (+ 1 2 3)
6
```

リストの最初の要素が関数の名前で、残りの要素が引数です。かけ算の関数*も使ってみましょう。

```
CL-USER> (* (+ 1 2) 3)
9
```

引数の計算を行なったのちに関数の呼び出しが行なわれます。この引数の計算のことを引数を 評価する と言います。

関数の定義 自分でも関数の定義を行なってみましょう。

```
CL-USER> (defun my-plus (x y)
  (+ x y))
MY-PLUS
CL-USER> (my-plus (* 2 3) 2)
8
```

2つの引数を足し算する関数が定義できました。

```
(defun <関数名> (<引数>*) [<省略可能なドキュメント文字列>] <本体の式>*)
```

最後の body form の結果が関数の返り値になります。

特殊オペレーター - special operator Common Lisp の構文はほぼ S 式しかありません。では条件分岐やループといったプログラムの制御構造はどうやって実現しているのでしょうか。

たとえば条件分岐を行なうために if という特殊オペレーターがあります。t は真を示したいときに慣習的に使用する値です。

```
(if <式> <条件が nil ではない> [<条件が nil>])
```

```
CL-USER> (if t (print "then") (print "else"))
"then"
"then"
CL-USER> (if nil (print "then") (print "else"))
"else"
"else"
```

Common Lisp では nil が偽でそれ以外は真です。if は関数で表現することはできません。もし関数であったとすると、

```
CL-USER> (defun my-if (p then else) (if p then else))
MY-IF
CL-USER> (my-if T (print "then") (print "else"))
"then"
"else"
"then"
```

というように、then の部分も else の部分も関数 my-if の引数ですから、両方とも評価された後に my-if が呼び出されてしまうのです。

長くなりそうなので詳しくは述べませんが、ループを実現できる機能としては C の goto のような動きをする go という特殊オペレーターがあります。

マクロの呼び出し Lisp では S 式で表現できる構文を自分でも定義することができます。それが Lisp のマクロです。

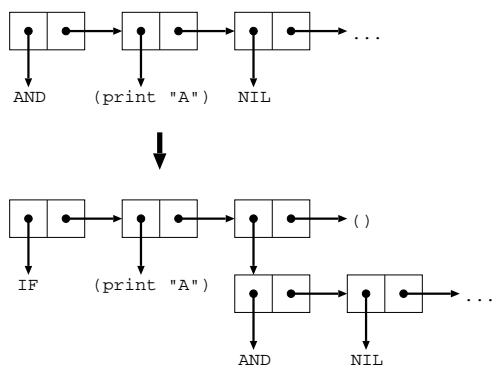
まずはもともと定義されているマクロ and を使ってみます。

```
CL-USER> (and (print "A") (print "B") (print "C"))
"A"
"B"
"C"
"C"
CL-USER> (and (print "A") nil (print "C"))
"A"
NIL
```

and マクロは引数の式の評価が真であるかぎりは残りを評価し、偽 (nil) より後は評価しません。最後に評価した値が結果になります。引数が無い場合は t が結果になります。マクロは S 式を S 式に変換する機能だと考えるとわかりやすいです。これはあるリスト構造を別のリスト構造に変換するということでもあります。関数 macroexpand-1 を使うと and マクロでどのような変換が行なわれたのかを見ることができます。

```
CL-USER> (macroexpand-1 '(and (print "A") nil (print "C")))
(IF (PRINT "A") (AND NIL (PRINT "C")) NIL)
T
```

引数の一つ目を条件とする if の式になりました。評価はマクロが全て変換された後に行なわれます。リストだと思って見てみれば以下のような変形です。



マクロの動きを理解するには S 式とリストを対応づけて見ていくのがコツです。

マクロの定義 自分でも and マクロのようなもの定義してみます。

```
CL-USER> (defmacro my-and (&rest forms)
  (if forms
      (list 'if (car forms) (cons 'my-and (cdr forms)))
      t))
MY-AND
CL-USER> (my-and (print "A") (print "B") (print "C"))

"A"
"B"
"C"
T
CL-USER> (my-and (print "A") NIL (print "C"))

"A"
NIL
CL-USER> (macroexpand-1 '(my-and (print "A") nil (print "C")))
(IF (PRINT "A") (MY-AND NIL (PRINT "C")))
```

car は cons セルから car を返す関数、list は複数の引数をリストにして返す関数、cons は 2 つの引数を car cdr の順に指す cons セルを作る関数です。&rest は可変引数をリストで受けとるためのパラメータの指定です。

このように、マクロはリスト構造を変換するようなプログラムを書いて定義を行ないます。マクロの定義の中身はマクロの展開のときに評価が行なわれるということが、ここでの重要なポイントです。

似たような動きをしているようですが、オリジナル and とは少し違っています。最後に評価されたものが結果にはなっていないようです。ここでは定義を単純にするために少し動きを変えてみました。

```
(defmacro <マクロの名前> (<引数>*) [<省略可能なドキュメント文字列>] <本体の式*>
```

11.2 Common Lisp のライブラリとマクロ

Common Lisp にも他の実用的な言語と同様に数多くのライブラリがあります。他の言語とは異なるかもしれない事情はライブラリ内のマクロの存在です。マクロの展開がすべて終わった後でないとプログラムをコンパイルし、実行することができないからです。

たとえば、あるライブラリ A は別のライブラリ B のマクロを使用しているかもしれません。すると、A のコードは B のマクロ定義をすべて展開した後でないとコンパイルすることができません。さらに、対話環境で開発を行なうことを考えたとき、利用することになっているライブラリをコンパイルが済んだ状態でロードしておきたいと思うかもしれません。そのときにはライブラリをロードした後にマクロ展開を全て行ない、その後にコンパイルする必要があります。数多くのライブラリを使用することにしていたら対話環境を利用できる状態にするまでに多くの時間がかかってしまいます。

このような状況を解決するために、Lisp の処理系では、マクロの展開とコンパイルが済んだ状態のイメージをダンプして保存しておいて再利用するのが一般的です。

11.2.1 ASDF - Another System Definition Facility

Common Lisp ライブラリのコンパイルを支援するためのライブラリとして ASDF があります。Makefile のようなもので、コンパイルに必要な情報を記述しておくことができます。ASDF ではライブラリモジュールのことを system と呼んでいて、system ごとに名前 (system name) を付けることになっています。同じモジュール内でファイル間に依存関係がある場合はそれを記述します。コンパイルに必要な別の system がある場合はその system name も記述します。Debian では cl-asdf パッケージです。以下は SBCL のドキュメントにあった ASDF のシステム定義の例です。

```
(defpackage hello-lisp-system
  (:use :common-lisp :asdf))

(in-package :hello-lisp-system)

(defsystem "hello-lisp"
  :description "hello-lisp: a sample Lisp system."
  :version "0.2"
  :author "Joe User <joe@example.com>"
  :licence "Public Domain"
  :components ((:file "packages")
               (:file "macros" :depends-on ("packages"))
               (:file "hello" :depends-on ("macros"))))
```

hello-lisp という名前のシステム定義を行なっています。

11.2.2 Common Lisp Controller

Common Lisp の処理系のダンプイメージを作りなおしてくれるツールです。処理系のパッケージを追加するとダンプイメージを作ってくれます。ライブラリをインストールすると各々の処理系ごとにダンプイメージを作り直してくれます。ライブラリが ASDF に対応していることが条件です。Debian では common-lisp-controller パッケージです。

11.2.3 dh-lisp

Common Lisp の処理系やライブラリの Debian パッケージ作成時に common-lisp-controller に対応させるための支援をしてくれるツールです。

パッケージのビルドの過程で dh_lisp コマンド呼び出すようにすると、パッケージ内の ASDF の定義を書いたファイル (.asd) を検索して、common-lisp-controller を呼び出すフックをメンテナスクリプトに追加してくれます。common-lisp-controller がメンテナスクリプトから呼びだされたときには asd ファイルの名前を見てダンプイメージ作り直しの対象かどうか調べてからダンプが行なわれます。/etc/common-lisp/images/<implementation>に asd ファイルの名前を書いておくと作り直しの対象になります。

現状だと例えばパッケージインストール用には以下のようなフックが追加されます。

```
if [ "$1" = "configure" ] &&
  which register-common-lisp-source > /dev/null; then
  register-common-lisp-source "#SYSTEMDIR#"
fi
```

#SYSTEMDIR# が asd ファイルの名前に置き換わります。

Common Lisp の処理系のパッケージを作成する場合には、ダンプイメージ出力のスクリプトを用意して、dh_lisp の引数に与える名前に合わせた名前を付けてやれば、やはり common-lisp-controller を呼び出すフックをメンテナスクリプトに追加してくれます。

現状だと例えばパッケージインストール用には以下のようなフックが追加されます。

```

case "$1" in
  configure)
    if [ -x /usr/lib/common-lisp/bin/"#IMPLEMENTATION#.sh" ] &&
      which register-common-lisp-implementation > /dev/null; then
      register-common-lisp-implementation "#IMPLEMENTATION#"
    fi
    ;;
  abort-upgrade|abort-remove|abort-deconfigure)
    if which register-common-lisp-implementation > /dev/null; then
      unregister-common-lisp-implementation "#IMPLEMENTATION#"
    fi
    ;;
esac

```

#IMPLEMENTATION#が dh_lisp の引数に与える名前に置き換わります。

11.3 Emacs での開発環境

最後に今回の話で使用している Emacs 上の対話環境の紹介をしておきます。

11.3.1 SLIME - Superior Lisp Interaction Mode for Emacs

SLIME は Emacs 用の Lisp 開発環境です。Debian では slime というパッケージに入っています。Emacs 側の Elisp で書かれたクライアントと Lisp の処理系側で書かれたサーバーが通信しながら対話的な開発環境を実現しています。Lisp の処理系側で書かれたサーバーの実装は swank と呼ばれています。swank を実装すれば他の Lisp の処理系でも slime を使うことができるらしいです。

Emacs からの利用方法ですが、たとえば処理系に SBCL を使用する場合は.emacs には以下のように書いておけばよいでしょう。

```

(setq slime-auto-connect 'ask)
(setq inferior-lisp-program "sbcl")

```

Emacs のキーバインドで、対話環境で試験的に実行してみるとときに良く使いそうなものを挙げておきます。

C-c C-z run-lisp Lisp 処理系との対話用バッファへスイッチ

C-c C-c slime-compile-defun カーソル位置の関数を対話用バッファの環境でコンパイル

C-c C-k slime-compile-and-load-file 編集中のプログラムのバッファのファイルを対話用バッファの環境でコンパイルしてロードする

C-c C-l slime-load-file 対話用バッファの環境で Lisp プログラムのファイルをロードする

11.3.2 Hyperspec

ANSI Common Lisp の仕様のオンラインドキュメントを SLIME から読むことができます。Debian では hyperspec というパッケージがインストーラーのパッケージになっています。

たとえば w3m-el パッケージを入れておいた状態で .emacs で

```

(set-default 'browse-url-browser-function 'w3m-browse-url)

```

などとやっておくと emacs バッファ内で関数やマクロのヘルプを読むことができます。次のキーバインドでドキュメント内の検索を行なうことができます。

C-c C-d h slime-hyperspec-lookup カーソル位置のワードで Hyperspec のドキュメント内を検索

11.4 参考文献

実践 Common Lisp

ISBN : 978-4274067211 / 著者 : Peter Seibel / 出版社 : オーム社

12 Debian on chumby の作り方

まえだこうへい



OSC 2009 Tokyo/Spring での東京エリア Debian 勉強会のブースで、Debian on chumby を行いました。今回はその環境の作り方についてまとめました。

12.1 概要

今回、実は chumby の上でネイティブに Debian を動かしたわけではありません。USB メモリにインストールした Debian に chroot して擬似的に動かしているように見せかけました。ネイティブに動かすとなるとブートローダをいじる必要がありますが、今回は chumby 自体はほとんど変更せずに済む方法をとりました。

12.1.1 chumby の仕様

chumby は、インターネットに接続可能な無線 LAN 環境が必要で、接続できなければアナログ時計の widget の表示しかできません。また、書き込み可能なメモリ領域はフラッシュメモリも 64MB のうち、わずかです*¹⁹。当然、Debian をローカルにインストールすることはできないので、USB メモリを外部ストレージとして使います。

もう一つの制約は、chumby は、ext2 などを使えません。USB メモリを使う場合は vfat のみです。しかし vfat では Linux をインストールできません*²⁰。そこで、大きく 3 つやる必要があります。

1. chumby のカーネルリビルド
2. USB メモリへの Debian インストール
3. USB メモリの Debian への chroot 設定

12.2 環境構築

12.2.1 前提条件

環境構築時に最低限必要なもの

- chumby
- USB メモリ
- Debian 環境構築用の PC
- ネットワーク環境

*¹⁹ jffs2 ファイルシステムで /psp としてマウントされています。

*²⁰ 原因は symlink を作成できないこと、適切なパーミッションを設定できないこと、など。

12.2.2 chumby のカーネルビルド

前述のとおり、chumby の kernel は ext2 を使えません。今回、USB メモリには ext2 フォーマットで Debian をインストールするので、chumby 自体も ext2 を読み込めるようにします。まず、下記リンク先から chumby のカーネル構築用のツールキットを入手します。手順はリンク先に従います。

- GNU Toolchain^{*21}
- GCC Toolchain^{*22}

これらのツールキットは、/usr 以下に展開されるので、kvm/qemu などの仮想 OS 環境に、環境を構築すると良いでしょう。

```
$ cd /
$ sudo tar zxf ~/arm-linux-v4.1.2b.tar.gz
$ sudo tar zxf ~/Gcc-3.3.2-glibc-2.3.2.tar.gz
$ sudo mkdir -p /opt/Embedix/usr/local/arm-linux
$ sudo ln -s /usr \
/opt/Embedix/usr/local/arm-linux/gcc-3.3.2-glibc-2.3.2
$ sudo vi /usr/bin/arm-linux-make
$ sudo chmod +x /usr/bin/arm-linux-make
```

/usr/bin/arm-linux/make には以下のように記述します。

```
#!/bin/sh
echo make ARCH=arm CROSS=arm-linux- CC=arm-linux-gcc \
AR=arm-linux-ar NM=arm-linux-nm RANLIB=arm-linux-ranlib \
CXX=arm-linux-g++ AS=arm-linux-as LD=arm-linux-ld \
STRIP=arm-linux-strip BUILDCC=gcc BUILD_CC=gcc \
CC_FOR_BUILD=gcc '$@'
exec make ARCH=arm CROSS=arm-linux- CC=arm-linux-gcc \
AR=arm-linux-ar NM=arm-linux-nm RANLIB=arm-linux-ranlib \
CXX=arm-linux-g++ AS=arm-linux-as LD=arm-linux-ld \
STRIP=arm-linux-strip BUILDCC=gcc
BUILD_CC=gcc CC_FOR_BUILD=gcc '$@'
```

私の chumby のファームウェアは 1.6^{*23} なので、Wiki の firmware 1.6 の手順を実施します。

- Hacking Linux for chumby - ChumbyWiki^{*24}

また、カーネルビルド用の環境には次の Debian パッケージは最低限入れておく必要があります。

- make
- gcc
- libncurses5-dev
- libncursesw5-dev
- zip

また、chumby 用のカーネルソースコードと、chumby へ新しいカーネルをインストールするためにアライメントする Perl スクリプトをそれぞれダウンロードしておきます。

- linux-2.6.16-chumby-1.6.0.tar.gz^{*25}
- align.pl^{*26}

カーネルソースを展開し、make menuconfig で ext2 を組み込みます^{*27}。

^{*21} http://wiki.chumby.com/mediawiki/index.php/GNU_Toolchain

^{*22} http://wiki.chumby.com/mediawiki/index.php/GCC_Toolchain

^{*23} 確認方法は、ssh で chumby にログイン後、chumby_version -f を実行します。

^{*24} http://wiki.chumby.com/mediawiki/index.php/Hacking_Linux_for_chumby

^{*25} <http://files.chumby.com/source/ironforge/build733/linux-2.6.16-chumby-1.6.0.tar.gz>

^{*26} <http://files.chumby.com/source/ironforge/build396/align.pl>

^{*27} モジュールにしても構いませんが、その場合は手動でカーネルモジュールをロードする必要があるので面倒です。

```

$ cd
$ mkdir kernel
$ cd kernel
$ cp ~/align.pl,linux-2.6.16-chumby-1.6.0.tar.gz ./
$ tar zxf linux-2.6.16-chumby-1.6.0.tar.gz
$ cd linux-2.6.16-chumby-1.6.0
$ ARCH=arm BOARD=mx21ads CROSS_COMPILE=arm-linux- \
make menuconfig
$ ARCH=arm BOARD=mx21ads CROSS_COMPILE=arm-linux- make
$ perl ../align.pl arch/arm/boot/zImage
$ zip k1.bin.zip arch/arm/boot/zImage

```

これで、kernel/linux-2.6.16-chumby-1.6.0/ディレクトリ直下に、k1.bin.zip が生成されます。これを USB メモリの vfat 領域にコピーします。

```

$ sudo mount -t vfat /dev/sda1 /media/usb
$ sudo mkdir /media/usb/update2
$ sudo cp -i k1.bin.zip /media/usb/update2/
$ sudo umount /media/usb

```

chumby を special option mode で起動し、kernel をアップデートします。

1. chumby の電源を OFF にした状態で USB メモリを挿します。
2. タッチスクリーンを押したまま、電源を入れる。途中で押したままにすると special option mode になるよ、と表示されるのでそのまま押しつづけます。
3. special option mode のメニュー画面で”install updates” をクリックします。
4. “Install from USB flash drive” をクリックすると、kernel がアップデートされ、自動的に再起動されます。

12.2.3 USB メモリへの Debian インストール

次に、USB メモリに Debian をインストールしますが、USB メモリには chumby 自体の設定を行うためのファイルを置く vfat 領域も必要なので、fdisk コマンドで/dev/sda1 を vfat、/dev/sda2 を Linux 用領域を作り、mkfs コマンドでファイルシステムを作成しておきます。

```

$ sudo fdisk /dev/sda
$ sudo mkfs.vfat /dev/sda1
$ sudo mke2fs /dev/sda2

```

この sda2 の方に、Debian をインストールします。chumby は、EABI(armel) ではなく、OABI(arm) であるため、arm 版のインストーラを用意する必要があります。しかし、Qemu では armel のサブアーキテクチャ versatile がサポートしていないため、arm 版の kernel イメージを起動させることしかできません。なので、今回は、同じ OABI のアットマークテクノ社の armadillo-9 用に公開されている Debian Etch イメージを利用しました。下記リンク先から、5 つの tar ボールを全てダウンロードします。

- [debian directory - Armadillo 開発者サイト](http://armadillo.atmark-techno.com/filebrowser/armadillo-9/debian)^{*28}

ext2 領域をマウントし、tar ボールを展開します。

```

$ sudo mount /dev/sda2 /mnt
$ cd /mnt
$ tar zxf ~/debian-etch-a9-1.tgz
$ tar zxf ~/debian-etch-a9-2.tgz
$ tar zxf ~/debian-etch-a9-3.tgz
$ tar zxf ~/debian-etch-a9-4.tgz
$ tar zxf ~/debian-etch-a9-5.tgz

```

chumby の電源を落とし、この USB メモリを挿して電源を入れると、自動的に ext2 領域もマウントされ、この下の領域のバイナリも正常に実行できます。

^{*28} <http://armadillo.atmark-techno.com/filebrowser/armadillo-9/debian>

12.2.4 USB 環境への chroot 準備

USB メモリの Debian 環境に chroot し、そしてその環境下で Etch から Lenny にバージョンアップさせます。まず、ssh でログインし、/proc、/dev、devpts をバインドさせます。

```
chumby:~# mount -o bind /proc /mnt/usb2/proc
chumby:~# mount -o bind /dev /mnt/usb2/dev
chumby:~# mount -t devpts devpts /mnt/usb2/dev/pts/
chumby:~# chroot /mnt/usb2
chumby:/1 df
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/hda1        1373548        181321  1118946  14% /
tmpfs            1373548        181321  1118946  14% /lib/init/rw
sysfs            1373548        181321  1118946  14% /sys
udev            1373548        181321  1118946  14% /dev
tmpfs            1373548        181321  1118946  14% /dev/shm
devpts          1373548        181321  1118946  14% /dev/pts
```

apt line を etch から lenny に書き換え、バージョンアップを行うと、問題なくアップグレードできるはずですが。

次に、chroot の Debian で、ssh を自動起動させるため、次の設定を行います。22/tcp は chumby 自体の sshd が使うので別のポートを割り当てる方が良いでしょう。

/mnt/usb2/etc/ssh/sshd_config (一部抜粋)

```
Port 2222
(snip)
PermitRootLogin no
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
(snip)
PermitEmptyPasswords no
ChallengeResponseAuthentication no
PasswordAuthentication no
(snip)
```

次に、chumby 側の設定。USB メモりに配置した Widget をロードさせる手順の応用で、6.2.3 で作成した vfat 領域の直下に、以下の内容で debugchumby というファイル名でスクリプトを作成します。

```
#!/bin/bash

mount -o bind /proc /mnt/usb2/proc
mount -o bind /dev /mnt/usb2/dev
mount -t devpts devpts /mnt/usb2/dev/pts/
chmod 666 /mnt/usb2/dev/null
chroot /mnt/usb2 /bin/hostname chumby
chroot /mnt/usb2 /usr/sbin/sshd
```

これで、次回以降、自動的に chroot 環境の Debian の sshd が 2222/tcp で起動するようになります。

12.3 OSC 会場での展示準備

6.1 でも書きましたが、Chummy はインターネットに繋がらないと単なる時計です。理由は、起動時に chummy.com から controlpanel.swf という管理コンソールの flash ファイルや、その他自分で設定している Widget をダウンロードしてくるためです。そこで、簡単な Widget を作成し、画面ではそれを表示しつつ、スタンドアロン環境でも有線 LAN を介して SSH でログインできるようにします。

12.3.1 前提条件

環境構築に加えて必要なもの

- mtasc パッケージ
- USB-Ethernet 変換アダプタ
- クロスケーブル
- 展示用の PC

12.3.2 Widget 作成

chumby の Widget は Flash です。Debian を使っているので Widget はもちろんテキストエディタで ActionScript を書いて、フリーソフトウェアでコンパイルします。今回の展示で表示させていた Widget のソースコードは以下のとおりです。

```
class DisplayDebian {
    public static function main(mc:MovieClip):Void
    {
        var app = new DisplayDebian(mc);
    }

    public function DisplayDebian(mc:MovieClip)
    {
        mc.createEmptyMovieClip('image',
            mc.getNextHighestDepth());
        var image:MovieClip = mc.image;
        var imageArr:Array = [ './openlogo.png' ];
        image._xscale= 100;
        image._yscale= 100;
        image._x= 69;
        image._y= 0;
        image.loadMovie(imageArr[0]);

        var textField:TextField = mc.createTextField('textField',
            mc.getNextHighestDepth(), 15, 10, 320, 240);
        var fmt:TextFormat = new TextFormat('', 24, 0x000000);
        textField.text = ' 東京エリア Debian 勉強会\n\n' +
            ' 次回は 3/21, 東大で開催予定';
        textField.setTextFormat(fmt);
    }
}
```

これを Hoge.as として保存し、同じディレクトリに openlogo.png^{*29}を配置します。
そして以下のワンライナー (ascompile.sh) の引数として渡し、コンパイルします。

```
$ ./ascompile.sh Hoge.as
```

ワンライナー ascompile.sh は以下のように記述します。

```
#!/bin/bash
test -z $1 && exit 1
mtasc -swf 'basename $1 .as'.swf -main $1 -header \
320:240:12 -version 8
```

コンパイルすると、Hoge.swf という flash ファイルができます。この Hoge.swf を Widget として読み込むために、profile.xml という名前で設定します。

^{*29} Debian.org のサイトのロゴ <http://www.debian.org/logos/openlogo.xcf.gz> を利用。そのままでは画像サイズが合わないため、gimp で高さ 240 ピクセルに収まるようにリサイズしています。

```

<?xml version='1.0' encoding='utf-8' ?>
<profile>
  <widget_instances>
    <widget_instance id='1'>
      <widget>
        <name>Debian Logo</name>
        <description>Debian GNU/Linux Logo</description>
        <version>1.0</version>
        <mode time='30' mode='timeout' />
        <access sendable='false' deletable='false'
          access='private' virtualable='false' />
        <user username='Kouhei Maeda' />
        <thumbnail href='file:///mnt/usb/openlogo.png'
          contenttype='image/png' />
        <movie href='file:///mnt/usb/Hoge.swf'
          contenttype='application/x-shockwave-flash' />
      </widget>
      <access access='private' />
      <mode time='30' mode='timeout' />
      <widget_parameters>
        <widget_parameter>
          <name>auther1</name>
          <value>Kouhei</value>
        </widget_parameter>
        <widget_parameter>
          <name>auther2</name>
          <value>Maeda</value>
        </widget_parameter>
      </widget_parameters>
    </widget_instance>
  </widget_instances>
</profile>

```

この profile.xml および、Hoge.swf と openlogo.png を USB メモリの vfat 領域直下にコピーします。これで、起動時に USB メモリからこの Widget が読み込まれるようになります。

12.3.3 スタンドアロンでの起動設定

次に、スタンドアロンで先ほどの Widget が読み込まれ、ssh でログインできるようにします。基本的にはフォーラム^{*30}の内容に従って行えば問題ありません。まず、chumby-offline.zip^{*31} をダウンロードします。展開すると offline-howto.txt というドキュメントがあるので、これに従い設定します。profile.xml は既に 6.3.2 で作成していますので、これを利用してください。USB メモリの vfat 領域の直下に以下のファイルをコピーします。

- chumby-offline.zip を展開してできる offline ディレクトリ以下にある www/ディレクトリ^{*32}
- chumby の /usr/widgets/controlpanel.swf

USB メモリの vfat 領域の直下の debugchumby を次のように書き換えます。

```

#!/bin/bash

killall httpd
/usr/sbin/httpd -h /mnt/usb/www
cp /mnt/usb/www/hosts.offline /psp/hosts
#cp /mnt/usb/www/hosts.online /psp/hosts

mount -o bind /proc /mnt/usb2/proc
mount -o bind /dev /mnt/usb2/dev
mount -t devpts devpts /mnt/usb2/dev/pts/
chmod 666 /mnt/usb2/dev/null
chroot /mnt/usb2 /bin/hostname chumby

```

なお、オフラインモードからオフラインモードに戻す場合は、killall から 3 行をコメントアウトし、hosts.online をコピーする行を有効にしてください。

そして最後に、USB-Ethernet アダプタを使い、有線 LAN 経由でアクセスできるように設定します。今までと同じく、vfat 領域に userhook1 というファイルを作成します。内容は以下のとおりです。

^{*30} <http://forum.chumby.com/viewtopic.php?pid=12258>

^{*31} <http://stud3.tuwien.ac.at/~e9825447/chumby-offline.zip>

^{*32} offline-howto.txt に従い、設定済みであること。

```
#!/bin/sh

USE_DHCP=0
IPADDR=192.168.3.10
NETMASK=255.255.255.0
GATEWAY=192.168.3.1

/sbin/insmod /drivers/usbnet.ko
/sbin/insmod /drivers/pegasus.ko

ifconfig rausb0 127.0.0.1

if [ $USE_DHCP == 1 ]\daggerhen
  udhcpc -t 5 -n -p /var/run/udhcpc.eth0.pid -i eth0
else
  /sbin/ifconfig eth0 $IPADDR netmask $NETMASK
  /sbin/route add default gw $GATEWAY eth0
fi
```

ちなみに、今回は BUFFALO の LUA2-TX を使っています。^{*33}

以上で、まったくインターネットを使えない環境でも、有線 LAN 経由で SSH 接続かつ、液晶モニタ上に USB に入れた Widget を稼働させることができるようになります。

12.4 残作業

以下が残っていますが、気が向いたらやるかもしれないし、やらないかもしれません。

- debootstrap で Lenny arm 版を作ってみて chroot 環境として使えるか？
- chroot 環境からフレームバッファを乗っ取る。
- ブートローダをハックして、直接 USB ブートさせる。

12.5 おまけ：今月のヨメ八苦

2月はこの OSC 準備と別件で、ヨメよりも早く帰ってきてても晩飯を作らなかった（作れなかった）のと、毎週水曜に Hack Meeting に参加するようになったため、めっちゃ不機嫌になりました。コワ。方々のアドバイスを頂き、3月現在は“すい〜つ”でなだめています。

12.6 参考文献

- chumby で遊ぼう！
 - 6.3.2 の Hoge.as は本書の p.48-49 の Main.as を、profile.xml は p.53 の USB メモリ用 profile.xml の例を、6.3.3 の userhook1 は p.156 の userhook1 を引用しました。（一部パラメータを変更）
 - ISBN : 978-4-7973-5039-5
 - 著者 : 米田 聡
 - 発行所 : ソフトバンククリエイティブ株式会社
- Using Chumby offline - Chumbysphere Forum
 - <http://forum.chumby.com/viewtopic.php?pid=12258>
- Hacking Linux for chumby - ChumbyWiki
 - http://wiki.chumby.com/mediawiki/index.php/Hacking_Linux_for_chumby

^{*33} これのデバイスドライバは pegasus.ko です。

13 Java ポリシーを読んでみた

まえだこうへい



13.1 Java ポリシーを読むハメになった経緯

初っ端からブチまけると、実は Java ポリシーなんて読むつもりは毛頭なかったのです。なぜなら私は Java なんて大嫌いだからです。長たらしいクラス名やメソッド名なんてみると、身の毛もよだちます。ちゃんと 80 バイトの幅に収めるよ、って感じです。(わら

冗談^{*34}はさておき、なぜ読むことになったかと言うと、実は別の目的がありました。典型的な手段が目的になってしまった例です。この発端は、友人と別件での活動で進捗管理が必要になったためです。某所の PC は admin 権限が与えられておらず、好き勝手にソフトウェアを入れられません。ですが、プロジェクト管理用に GanttProject^{*35}はライセンス使用料無しで導入できるので、これで管理するか、ということにしました。オープンソースソフトウェアなので、きっと Debian にもパッケージがあるに違いない、と。WBS を作り、自宅で Debian に GanttProject を導入しようとしたら、パッケージが無いではありませんか。使えないとせっかく作った WBS も更新できず困ります。そこで仕方ないので、ITP してみることにしました。ちょうど、まだ BTS へ投げたことも、ITP もしたことがなく、ある意味ちょうど良いきっかけなので ITP してみました^{*36}。

で、Debian Hack Cafe で、岩松さんから「どうせなら Java Policy 読んで、勉強会のネタにする」と言われた訳です。特に断る理由もなかったので、やることにしてみました。が、ちょうど今回の Debian 勉強会と、KVM の記事^{*37}の公開が時期的に重なってしまったため、締切り直前で勉強会資料の作成をやっています。終わるのかな。当日の事前配布資料に、このネタが掲載されていればきっと間に合ったのでしょう。

ちなみに、肝心の当初の目的ですが、完全に置いてけぼりを喰らっています。Java ポリシーの翻訳はもう少しで終わるのですが、ITP した GanttProject の deb パッケージ化は未着手、GanttProject で進捗管理する対象は友人が中心に進めているもの、進捗は遅れ気味になっています。巻き返ししなければ。

13.2 さて、今回の本題。

前置きが長くなりましたが、今回の本題は、Java ポリシーについて調査してみたよ、でした。要約によれば、背景説明、ポリシーの内容、議論すべき問題点、Java パッケージメンテナ向けのアドバイスであり、Java 仮想マシン、Java コンパイラ、Java プログラム、Java ライブラリをターゲットにしています。以下、各章ごとに要点をまとめてみました。

*34 嫌いなのは本当なんです。

*35 <http://www.ganttproject.biz/>

*36 <http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=436792>

*37 <http://www.atmarkit.co.jp/flinux/rensai/kvm02/kvm02a.html>

13.3 1章 背景説明

要点は2つです。

- 特定の議題をより詳細に決める場合、Debian ポリシーにはサブポリシーとして扱われる。
- コメントなどをするには、java-common パッケージチームや、Debian Java メーリングリストに投稿する。

要は、Java には固有の事情があるので、Java ポリシーというサブポリシーで扱っているよ、ということです。

13.4 2章 ポリシー

Java パッケージに対するポリシーをまとめています。そのポリシーとは以下のものです。

- コンパイラ、仮想マシン、ランタイム用の仮想パッケージが作られます。
- 全 Java コードは、Java バイトコードとして、全アーキテクチャ向けに配布しなければなりません。
- 開発者向け (ライブラリ) とエンドユーザ向け (プログラム) の2つのカテゴリに分類されます。

13.4.1 仮想マシン

Java 仮想マシンをパッケージ化する際のポリシーです。パッケージ名、依存関係、コマンド名を規定しています。

- java-virtual-machine パッケージを提供し、java-common パッケージに依存していなければなりません。
- ランタイム環境を配布することもできます。
- Java2 に準拠したランタイムをパッケージでは、java2-runtime を提供すべきです。^{*38}
- Sun Java プログラムと互換性のあるコマンドラインであれば、`/etc/alternatives/java` という名前にすべきです。
- 必要なランタイム環境を、あらかじめ CLASSPATH に設定しておくべきです。
- JDK のようにコンパイラや仮想マシンのソースコードを提供する場合は、コンパイラパッケージを `xxxx-dev` と命名しなさい。
- Java クラスの実装は、ネイティブ言語を用いた処理系で、実行時にロードされる動的ライブラリとしてコンパイルされ、配置されています。仮想マシンがネイティブコードをサポートする場合は、動的ライブラリの検索パスに `/usr/lib/jni` ディレクトリを設定しないとけません。

13.4.2 コンパイラ

Java コンパイラに関するポリシーです。仮想マシンの時と基本的には同じで、3点挙げられています。

- java2-compiler パッケージを提供し、java-common パッケージに依存していなければなりません。
- Sun JDK の javac と互換性があるなら、`/etc/alternatives/javac` という名前にすべきです。
- コンパイルに必要な java コアクラスを CLASSPATH に設定しておくべきです。

13.4.3 Java プログラム

Java のプログラムに関するポリシーです。

- `/usr/bin` 以下に配置し、実行可能でなければなりません。
- プログラムは `binfmt_misc` を使用した Java クラスか、あるいはラッパーでも構いません。

^{*38} Java1.1 の場合は、`java1-runtime` なのですが、今時 Java1.1 はないと思われるので省略。

- 特定の環境変数^{*39}がなくても、実行できなければなりません。
- 実行ファイルに関するポリシールールに従う必要があります。
- プログラム自体が補助クラスの場合は、`/usr/share/java` ディレクトリ配下に `jar` ファイルとして配置しなければなりません。
- プログラム自体は仮想マシン (`java-virtual-machine`) とランタイム環境 (`java2-runtime`) に依存していなければなりません。
- プログラムの名前規則は無く、ユーザ視点で一般的なプログラムです。

13.4.4 Java ライブラリ

Java ライブラリに関するポリシーです。Java ならではの特徴的なことは、開発者向け^{*40}やユーザ向けのバージョンには分かれていないという点です。

- Java では開発者向けとユーザ向けとでライブラリのバージョンを分けません。意味がないからです。
- Java ライブラリパッケージは、`libXXX[version]-java` という形式の名前でなければなりません。
- `version` 部分はオプションで、必要な部分だけを含むべきです。
- `version` 部分は名前の衝突を避けるようにすべきです。
- `XXX` は実際のパッケージ名です。
- `jar` アーカイブのクラスは、`/usr/share/java` 配下に配置しなければなりません。
- 名前は `package[-extraname]-fullversion.jar` の形式です。
- `extraname` はオプションで、パッケージで提供されている `jar` とは別に分けるためにパッケージ内部で使われます。
- `fullversion` は `jar` ファイルのバージョンです。パッケージとは同一ではない場合があります。
- Java ライブラリはランタイム環境には依存しなければなりません。仮想マシンには依存すべきではありません。

など。

13.4.5 main, contrib, non-free

`main`, `contrib`, `non-free` に分類するためのポリシーです。

- ソースパッケージが `non-free` なツールでしか (正しく) コンパイルできないなら^{*41}、`main` にすることはできません。パッケージ自体が自由であるなら、`contrib` に入れなければなりません。
- バイナリパッケージが `non-free` の仮想マシンだけで動くことができるなら^{*42}、`main` にすることはできません。パッケージ自体が自由であるなら、`contrib` にしなければなりません。

13.5 議論すべき問題

現在、8 つほどの問題が取り上げられています。いずれも議論の余地があるとのこと。

- リポジトリの名前と存在について。最新バージョンでは削除されています。
- `/usr/share/java` におけるシンボリックリンクを C 言語のライブラリのようにスクリプトで置き換えるべきか否か。
- コアクラスの必要性について。

^{*39} 例えば、`CLASSPATH` など

^{*40} `-dev` がパッケージ名につくもの。

^{*41} 自由な Java コンパイラは、`guavac` と、`gcj` と `jikes` だけのようです

^{*42} `GNU-Classpath` には、自由なバージョンがリストされています。

- Sun のコミュニティーソースライセンスは、Debian では使えるのか？使う場合はどうすれば良いのか
item すべての jar ファイルは適切な CLASSPATH ドキュメントが必要です。

以下、省略。

13.6 Java パッケージ作成者向けのアドバイス

Java パッケージ作成者向けのアドバイス、とはなっているものの、実際のところは、足りないツールを作ってくれだとか、他の言語向けパッケージ作成では自動で行っている部分を手動でやれ、といった話になっています。しかもひどいことに、「これはアドバイスに過ぎず、ポリシーの一部ではない」と言っています。

- debian/control のすべての依存関係を手動で管理しなければなりません。Debian の開発ツールでは Java のプログラムを検出することができません。
- だから、Perl のように dh_java プログラムを作ってくれるボランティアは大歓迎とのこと。
- debian/rules の中の、dh_strip や dh_shlibdeps のような Java には無意味なコマンドを無効にしましょう。

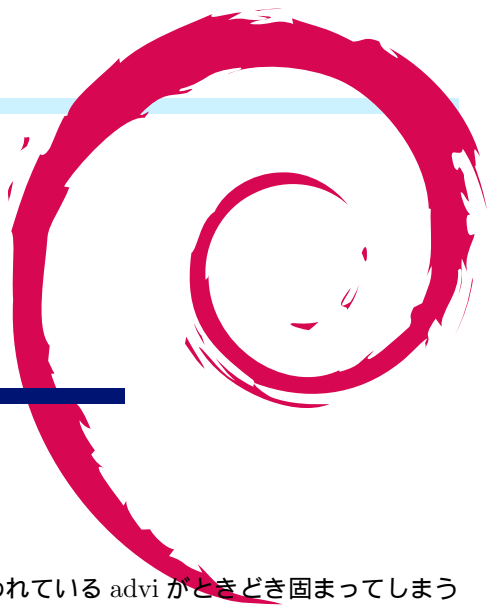
などです。

13.7 まとめ

実は、ポリシーマニュアルをちゃんと読んだことがまだないので、通常のポリシーに比べた場合の、Java ポリシーの特徴がどうなのかがよく分かりません。ただ、自由なコンパイラが 3 つあるとか、main に入れるための制約とか、命名規約やら、依存関係についてのポリシーがあることを知りました。これを元に途中で止まっていた ganttproject の deb パッケージ化を進めます。またプロセスは、本来の目的を達成するために早く進めなきゃです。ついでに、今回訳した文章は、取り合えず debian-doc に投げて査読してもらおうと思います。

13.8 今月のヨメ八苦

最近、毎週水曜の Hack Cafe は認めてもらえるようになりました。まあ、本人もその時間にジムに行けるからという理由もありますが。しかし、IRC 会議や Debian 勉強会以外で、Debian 絡みの作業したり、外にでかけると、「また Debian？」と怪訝な顔をされます…。なかなか思うとおりにはならんものです。



14 advi をデバッグしてみた

日比野 啓

2008 年 11 月の LaTeX を使ったハンズオンで、wizzytex-mode から使われている advi がときどき固まってしまう問題について調べてみました。

14.1 advi が固まる?

advic は一見普通の DVI viewer なのですが、なぜか OCaml という変わった言語で実装されています。今回は advic から呼ばれる ghostscript が止まっているらしい、ということまで分かっている状態から調べ始めました。

14.2 とりあえずアタリをつける

とりあえず、問題が起きているソースを取ってきて展開してみます。

```
% apt-get source advi
...
dpkg-source: extracting advi in advi-1.6.0
dpkg-source: info: unpacking advi_1.6.0.orig.tar.gz
dpkg-source: info: applying advi_1.6.0-13.diff.gz
% cd advi-1.6.0
% ls *.ml
addons.ml      drawimage.ml  font.ml       gs.ml         main.ml       search.ml     transimpl.ml
ageometry.ml  driver.ml     global_options.ml  gterm.ml     misc.ml       shot.ml       ttfont.ml
...
```

*.ml というのが OCaml のソースファイルです。なんか、gs.ml とかいうそのものズバリっぽいものが見えます。gs.ml の中をまず gs で検索していってみると、

```
...
let command = Config.gs_path in
let command_args =
  [
    command;
    "--dNOPLATFONTS"; "--dNOPAUSE";
    "--sDEVICE=" ^ (if !antialias then x11alpha else x11);
    "-q";
    "--dSAFER";
    "-";
  ] in

let _ = debugs command;
...
```

おお、それっぽい。あと、デバッグ用っぽい機能 - debugs を発見。さらにこんどは command で探していくと、

```

...
let lpd_in, lpd_out = Unix.pipe () in
...
let leftout = Unix.out_channel_of_descr lpd_out in
...
let pid =
  Unix.create_process command command_args lpd_in rpd_out
  (* Unix.stdout *) Unix.stderr
...
method line 1 =
  try
    showps l;
    output_string leftout l;
    output_char leftout '\n';
  ...

```

どうやら gs にパイプで PS を書きこんでいるようです。showps とかいうので PS の中身を見ることができんじゃないかなーとか。

14.3 まじめに調べてみたんですが...

もう一度、こんどは gs.ml の最初の方からデバッグ用の機能だけ見ていきます。

```

...
let debugs = Misc.debug_endline;;
...
let showps_ref = ref false;;
let showps s =
  if !showps_ref then (print_endline (Printf.sprintf "%s" s));;
...
Options.add
  "--showps" (Arg.Set showps_ref)
  " ask advi to print to stdout a copy\
  \n\t of the PostScript program sent to gs.>";;
...

```

Misc. というのは Misc という別のモジュールへの参照です。ここでは単に misc.ml の中を見ればよさそうです。showps_ref は書き換え可能なフラグのようです。と思ったらすぐ下にコマンドライン引数からフラグをセットできるようになっているようです。misc.ml の中も見ると、

```

...
(* Debugging. *)
let forward_debug_endline =
  ref (function (_ : string) -> failwith "undefined forward debug_endline");;

let debug_endline s = (!forward_debug_endline s : unit);;

let set_forward_debug_endline f = forward_debug_endline := f;;
...

```

さらに set_forward_debug_endline で grep すると、global_options.ml が引っかかるので、その中も見ると

```

...
(* To print debugging messages. *)
let debug_endline = Options.debug "--debug" " General debug";;

(* Setting the forward in Misc. *)
Misc.set_forward_debug_endline debug_endline;;
...

```

結局、どちらもコマンドラインから設定できるようです。さっそく試してみると、

```

% platex debianmeetingresume200812-presentation.tex
...
% advi debianmeetingresume200812-presentation.dvi
...
/usr/bin/gs
-dNOPLATFONTS
-dNOPAUSE
-sDEVICE=x11
-q
-dDELAYSAFER
-
...
%!PS-Adobe-2.0
%%Creator: Active-DVI
%!
[1 0 0 -1 0 0] concat
(/usr/share/texmf-texlive/dvips/base/texc.pro) run
(/usr/share/texmf-texlive/dvips/base/special.pro) run
...
%% Newpage

grestore
0 0 moveto
TeXDict begin 12769384 12769384 div dup /Resolution X /VResolution X end
TeXDict begin /DVImag 194.845342 def end
gsave
flushpage (...
) print flush

```

たしかに gs のコマンドラインらしきものと、それから書きこんだ PS の内容らしいものが見えています。PS で目印となる文字列を出力される命令 flushpage (...) print flush を gs に書きこんで、その出力を待っているようなのですが、戻ってきていないようです。gs が止まってしまう場合とそうでない場合も比べてみたのですが、止まってしまう場合の PS の最小セットを割り出すのが難しく、よくわかりませんでした。

14.4 別の回避策?

なにか別の方法で止まってしまうのを回避できないか、と gs の出力を待っている部分も見てみます。

```

...
let rec select fd_in fd_out fd_exn timeout =
(* dirty hack: Graphics uses itimer internally! *)
let start = Unix.gettimeofday () in
try
Unix.select fd_in fd_out fd_exn timeout
with
Unix.Unix_error (Unix.EINTR, _, _) as exn ->
let now = Unix.gettimeofday () in
let remaining = start +. timeout -. now in
if remaining > 0.0 then select fd_in fd_out fd_exn timeout else [], [], []
...
match select [ rpd_in ] [] [] 1.0 with
| [], _, _ ->
begin match Unix.waitpid [ Unix.WNOHANG ] pid with
| x, Unix.WEXITED y when x > 0 ->
raise (Killed "gs exited")
| 0, _ ->
raise (Killed "gs alive but not responding")
| _, _ ->
raise (Killed "gs in strange state")
end
...

```

gs の出力を select で待っているようです。タイムアウトも仕込んであるようです。なぜうまくいっていないのでしょうか。

ここでは前半で定義されている select に注目です。せっかくタイムアウトの残り時間を計算しているのに、渡しているのはもとの値です。どうりでいつまでたってもタイムアウトしないわけです。

```

...
if remaining > 0.0 then select fd_in fd_out fd_exn timeout else [], [], []
...

```

これを

```

...
if remaining > 0.0 then select fd_in fd_out fd_exn remaining else [], [], []
...

```

と直すと、gs を待ってもタイムアウトするようになります。gs が固まる原因を取り除くような根本的な解決はできませんでしたが、とりあえずは advi が止まらないようにはなりそうです。



15 ocaml に入門してみた

上川純一

15.1 ocaml 入門の動機

ocaml はフランス圏で流行っている言語のようです。advi や unison などのかゆいところに手が届く感じのソフトウェアが ocaml で実装されていて、気にはなっていました。ocaml は素晴らしい計算機理論を活用して最新の何かの実装されているという噂は聞いているのですが、いまいち何がどうすごいのかよくわからないのでものは試しと、試してみました。

15.2 ocaml を使うために

Debian GNU/Linux には ocaml 関連のパッケージが多数入っています。ためしに検索してみると 224 パッケージありました。

```
$ apt-cache search ocaml | wc -l
224
```

ocaml にはインタラクティブに使うためのインタプリタと、バイナリを生成してくれるコンパイラの両方があります。ocaml をとりあえず一通り使うためには ocaml のインタプリタとコンパイラがあればよさそうです。ocaml のネイティブコード用のコンパイラは ocaml-native-compilers パッケージのようです。ocaml のインタプリタは ocaml-interp パッケージのようです。

emacs ユーザとしては、emacs 用のモードも必要です。emacs のモードでそのものずばりの ocaml-mode というものもあるのですが、tuareg というのがよさそうなので、それをインストールしてみました。

```
# apt-get install tuareg-mode ocaml-native-compilers ocaml-interp ocaml
```

15.3 インタラクティブに使ってみる

インタラクティブに ocaml を使ってみました。まず足し算をしてみます。

emacs から M-x tuareg-run-caml で ocaml のインタラクティブな端末 (ocaml 用語では「トップレベル」という) を起動しました。足し算をするには、`1 + 2` を入力し、文の終了は `;;` で表現します。

```
# 1 + 2 ;;
- : int = 3
```

int 型の 3 がかえってきました。めでたしめでたし。

15.4 関数を定義してみる

では、関数を定義してみます。

```
# let my_func a b = a + b ;;
val my_func : int -> int -> int = <fun>
# my_func 2 3;;
- : int = 5
```

int int をとって int をかえす関数が定義され、それを 2 と 3 で呼び出したら 5 が帰って来ました。めでたしめでたし。

型をまったく宣言していないですが、勝手に定義されているあたりが型推論機能のようです。どうして型が推論できるかということに疑問に思われるかもしれませんが、「+」は実は int 専用の足し算です、float の場合は「+.」を使うそうです。これを知って失神しそうになりました。

15.5 コンパイルしてみる

とりあえずインタラクティブに使う方法はわかったので、コンパイルして使う方法を見てみましょう。ocaml は中間言語に変換する方法とネイティブコンパイルする方法の二つがあるようです。ocamlc は中間言語、ocamlopt はネイティブコンパイラのようなので。ocamlc には二種類のバイナリが提供されており、ocamlc と ocamlc.opt があります。ややこしいですが、ocamlc.opt はネイティブコンパイラでコンパイルした中間言語コンパイラのようなので。

```
$ time ocamlc fibo.ml -o a.1
real    0m0.038s
user    0m0.032s
sys     0m0.004s
$ time ocamlc.opt fibo.ml -o a.2
real    0m0.014s
user    0m0.012s
sys     0m0.000s
```

種類を表 2 にまとめてみました。

表 2 ocaml コンパイラの種類

	ネイティブコンパイルされた	中間言語にコンパイルされている
ネイティブコンパイル結果を出力	ocamlopt.opt	ocamlopt
中間言語を出力	ocamlc.opt	ocamlc

簡単な fibonacci のコードを書いてみて動作速度の雰囲気を感じてみました。バイトコードのオーバーヘッドが感じられる結果になりました。

```
$ time ./byte-code
102334155
real    0m14.544s
user    0m14.541s
sys     0m0.000s
$ time a./native-code
102334155
real    0m2.933s
user    0m2.932s
sys     0m0.000s
$ time ./fibonacci.c 40
102334155
real    0m2.325s
user    0m2.192s
sys     0m0.040s
```

15.6 ocaml 関連のパッケージ化の課題

Debian パッケージにする際には、コンパイルする回数よりもインストールして実行される回数のほうが多いため、基本としてはネイティブコンパイルを選択します。ただし、ネイティブコンパイルできない CPU アーキテクチャもあるため、どのコンパイラを利用するのかを抽象化するレイヤーが必要です。

ocaml packaging policy によると、ネイティブコンパイラに対応しているのは amd64, i386, kfreebsd-i386, powerpc, sparc だけだそうです。重要なアーキテクチャで動きません。たとえば、arm、superh、mips などには対応してないようです。

ライブラリの扱いは特殊で、`/usr/lib/ocaml/3.10.2/`以下においてあるようです。ライブラリといっても、バイナリの `libxxx.so` 共有ライブラリがあるわけではなく、`.mli` インタフェースファイルが大量にあります。ocamlc `-where` で発見できるようです。

```
$ ocamlc -where
/usr/lib/ocaml/3.10.2
```

ひどい点は、年に一回くらいマイナーバージョンがリリースされているのですが、その度にライブラリコードがバイナリ非互換に変更されているっぽく、リビルドするハメになっているようです。

Debian の ocaml メンテナンスチームはあらゆるパッケージを一つの subversion リポジトリに統合して管理しているようですが、その理由の一つがこのバージョン間の非互換性ではないでしょうか。

15.7 参考

- OCaml packaging policy http://pkg-ocaml-maint.alioth.debian.org/ocaml_packaging_policy.txt
- dh-ocaml パッケージ

16 Erlang コードを Debian パッケージ にしてみた

まえだ こうへい



16.1 入門のきっかけ

Erlang というプログラミング言語があります。今回、これを勉強してみようと思ったのは、Erlang そのものがきっかけではなく、いつものように別のものがきっかけです。今回は、CouchDB という Apache Incubator^{*43}にあった^{*44}、ドキュメント指向のデータベースを友人から教えてもらったのがきっかけでした。CouchDB が Erlang と JavaScript で実装されている、というので、「Erlangって何だろう？」と興味を持ちました。

16.1.1 ちょっと脱線：CouchDB とは

CouchDB の特徴には以下のものがあります。

- ドキュメント指向
- スキーマレス
- RESTful
- JSON 形式でデータをやりとり

RDB のように事前にしっかりとスキーマの定義を行っておく必要はなく、HTTP PUT でデータ (= ドキュメント) を作成・更新し、GET でデータを取得し、DELETE でデータを削除する、ちょっと変わったデータベースです。RDB を置き換えるものではないので、摘要範囲は正直まだよく分かりません。ですが、面白そうなので友人とこれを広めるべく、目下水面下で活動中です。

CouchDB のアーキテクチャについては、CouchDB Project のホームページから転載した、下記のシステム概念図をご覧ください。これらの実装は主に Erlang によるものです。

また、CouchDB については、水面下の活動の一環で翻訳なども行っているのですが、また別の機会にお話できればと思います。

16.1.2 閑話休題：Erlang の概要

本題の Erlang ですが、CouchDB に興味を持ち、さらにその中身をどうなっているのを知りたいと思ったのですが、Erlang という言語を知りません。関数型プログラミングで、並列処理、分散処理、耐障害性を兼ね備えた、並行指向プログラミングというのですが、そもそも文法がまったく分からないので、ソースコードだけを眺めて理解できるものではないな、ということから勉強を始めてみました。また、並列処理、分散処理に興味を持っていたので、興

^{*43} <http://incubator.apache.org/>

^{*44} 現在は、Apache Incubator から卒業して Apache Project の一つになっています。 <http://couchdb.apache.org/>

味がうまい具合に重なったのも、きっかけとなりました。

Erlang の歴史

Erlang は、Ericsson Computer Science Laboratory で、分散化された環境、耐障害、ある程度のリアルタイム性、無停止で稼働する、といった条件のシステムを構築できるように設計されたプログラミング言語です。もともとは、エリクソンの社内だけで使われていましたが、1998 年にオープンソースとして公開されました。ライセンスは、Erlang Public License^{*45}で、Mozilla Public License の派生ライセンスです。関数型で並列指向のプログラミング言語ということですが、20 年を超える歴史と、高度な信頼性が要求される通信機器の分野でも実績があり、例えばエリクソンの AXD301^{*46}などがあるそうです。

仕様

プログラミング Erlang という書籍を一通り読んで、今 2 周目で実際に手を動かしている段階ですが、その中で興味を持った特徴を挙げます。

- 評価できるものは全て式
- 変数の書き換えはできない
- 関数型プログラミング言語
- 並行処理 (並行プログラミング)

自分には馴染みが薄いものばかりです。特に、「変数の書き換えはできない」なんて、定数じゃないの? と思いました。詳しくは後述。

Erlang による最近の実装例

Erlang で実装されているソフトウェアの例です。これらは deb パッケージになっており、Lenny にも含まれています。

- CouchDB
- ejabberd^{*47} twitter で使われているという IM サーバ
- YAWS(Yet another web server)^{*48} Erlang で実装された Web サーバ

Erlang の検証事例

数年前から、一部では流行りらしいので、検証事例もちらほら出ています。

- KLab 株式会社による検証
 - Erlang Performance
http://lab.klab.org/wiki/Erlang_Performance
 - Erlang Process
http://lab.klab.org/wiki/Erlang_Process
- Erlang vs. Stackless python: a first benchmark
<http://muharem.wordpress.com/2007/07/31/erlang-vs-stackless-python-a-first-benchmark/>

一般的に普及していくのはまだこれからみたいな感じしますが、あまり使われていないうちに遊んでおこうと思います。

^{*45} <http://erlang.org/EPLICENSE>

^{*46} ATM スイッチ

^{*47} <http://ejabberd.jabber.ru/>

^{*48} <http://yaws.hyber.org/>

16.2 環境を整える

Debian では数はまだ少ないものの Erlang と Erlang で書かれたソフトウェアのパッケージがあります。

```
$ apt-cache search erlang | wc -l
63
```

ちなみに、例として挙げた CouchDB や YAWS も deb パッケージになっています。

Erlang を使うために最低限必要なパッケージは次の erlang-base パッケージです。対話型ではなく、ソースコードを書くのには emacs の Erlang 用のモードもあるので、それも一緒にインストールしておくといいでしょう。

```
$ sudo apt-get install erlang-base erlang-mode
```

ところで、erlang-base パッケージではなく、erlang パッケージをインストールすれば、erlang ソースパッケージから分割されている関連パッケージも軒並みインストールされるので便利かもしれませんが、この場合は依存関係も含めて関連するパッケージが 90 以上導入されます。状況に応じて適宜選択してください。

```
$ sudo apt-get install erlang
```

16.3 対話型で試してみる

対話型で Erlang を使うには、erl コマンドを実行します。すると、Erlang shell(Eshell) が起動します。

```
$ erl
Erlang (BEAM) emulator version 5.6.5 [source] [64-bit] [smp:2] [async-threads:0] [kernel-poll:false]

Eshell V5.6.5 (abort with ^G)
1>
```

“1>” はプロンプトです。Eshell を終了するには、q(). と入力するか、

```
1> q().
ok
2> $
```

または、ctrl+ c, a と入力します。

```
1>
BREAK: (a)abort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
a
$
```

では、変数を書き換えられない、という件の特徴をみてみます。

```
1> A=10.
10
2> A=20.
** exception error: no match of right hand side value 20
```

見ての通り、エラーになってしまいました。Erlang の変数の書き換えはできない、単一代入変数なのです。もう一度代入の仕方を変えて見てみましょう。

```

1> A1=1.
1
2> A2=2.
2
3> A3=A1+A2.
3
4> A3=3.
3
5> A3=A1+2.
3
6> A3=A1+4.
** exception error: no match of right hand side value 4
7> A1=A2+A3.
** exception error: no match of right hand side value 5
8> A1=A3-A2.
1

```

1,2,3 のように未代入の変数に対しては値を再代入できますが、6,7 のように一度代入された変数に対しては違う値を代入しなおすことができないことが分かります。しかし、4,5,8 のように同じ値であれば再代入できるように見えます。実は、4,5,8 は変数を再代入しているのではなく、代入済みの変数に対し、パターン照合を行っているのです。

代入済みの変数のことを束縛済み変数、と言いますが、未束縛状態の変数に対しては、イコールは値の代入を行いますが、束縛済みの変数に対してはパターン照合を行う、という訳です。

今までこんな仕様のプログラミング言語は見たことなかったのですが、プログラミング言語ではなく、実は数学の代数であれば同じような考え方ができることが分かります。例えば、 $x + y = 6$, $x - y = 2$ という関数を考えてみます。この状態では、 x , y の代数は $x=4$, $y=2$ 以外の値にはなりません。単一代入変数は、数学の代数と同じようなものと考えれば、良いのかもしれませんが。

16.4 コンパイルしてみる

次に、ソースコードからコンパイルして実行する方法について説明します。プログラムを実行する方法は、次の3種類があります。

- Eshell からコンパイルして実行する。
- シェルからコンパイルして実行する。
- escript として実行する。

例としてはやはり、Hello World を書いてみます。これは引数なしのプログラムです。引数ありとなしではプログラム実行時に若干の違いが出てきます。hello.erl というファイル名で次のプログラムを用意して下さい。

```

-module(hello).
-export([start/0]).

start() ->
  io:format("Hello world on Erlang~n").

```

Erlang のプログラムはモジュールを基本単位とし、どの関数もモジュールに属する構造になっています。コードを実行するには、まずモジュールをコンパイルしなければなりません。コンパイルしたモジュールは、拡張子 beam^{*49} という名前のファイルが生成されます。コンパイルには2つのやり方があります。

16.4.1 Eshell からコンパイルする方法

先ほどの hello.erl を Eshell からコンパイルし、実行するには、次の手順です。

```

1> c(hello).
{ok,hello}
2> he
heart    hello
2> hello:start().
Hello world on Erlang
ok
3>

```

^{*49} Bogdan's Erlang Abstract Machine の略。

c(hello). でコンパイルの実行を行います。c() の引数が module(hello) および、プログラムファイル名 hello.erl と一致することに注意してください。コンパイルすると、hello.beam ファイルが生成されます。これは Erlang VM 向けのバイナリコードです。

```
$ ls -lrt hello.*
-rwx----- 1 user user  90 2009-05-14 01:35 hello.erl
-rw----- 1 user user 552 2009-05-14 01:35 hello.beam
$ file hello.beam
hello.beam: Erlang BEAM file
```

16.4.2 シェルからコンパイルする

シェルからコンパイルする場合は、erlc コマンドを使います。

```
$ erlc hello.erl
$ ls -lrt hello.*
-rwx----- 1 user user  90 2009-05-14 01:35 hello.erl
-rw----- 1 user user 708 2009-05-14 01:53 hello.beam
```

実行もシェルから行ってみます。

```
$ erl -noshell -s hello start -s init stop
Hello world on Erlang
```

無事、実行できました。なお、各引数は次の意味があります。

- -noshell
対話型シェルなしで Erlang を起動する。
- -hello start
関数 hello:start() を実行する。
- -init stop
apply(hello, start, []) が完了すると^{*50}、システムは関数 init:stop() を評価する。

16.4.3 escript で実行する

Erlang では escript という仕組みもあり、これを使うとプログラムをコンパイルせずにスクリプトとして直接実行できます。

関数 hello:start を escript として実行するには、次のようにファイルを書き換えます。ファイル名は今回、hello.esc としましたが、コンパイルするときと違い、任意の拡張子で問題ありません。

```
#!/usr/bin/env escript
main(_) ->
  io:format("Hello world on Erlang\n").
```

これを実行権限をつけて実行してみます。

```
$ chmod 700 hello.esc
$ ./hello.esc
Hello world on Erlang
```

16.4.4 引数ありの関数をコンパイル、実行する

今回の例は、引数なしでしたが、シェルで実行する場合のプログラムはどうやって引数を渡せば良いのでしょうか。今度は引数ありのプログラムを試してみます。

フィボナッチ数列を計算する関数を書いてみました。まず、Eshell で引数を渡す場合です。

^{*50} -s ... コマンドは apply 文で評価される。1 つのコマンドが完了すると次の -s ... コマンドが評価される。

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 1;
fib(N) ->
    fib(N-2) + fib(N-1).
```

実行するとこんな感じ。

```
1> c(fib)
1> .
{ok,fib}
2> fib:fib(10).
89
3> fib:fib(20).
10946
4> fib:fib(30).
1346269
```

シェルから実行する場合は、書き方を変える必要があります。

```
-module(fib1).
-export([main/1]).

main([A]) ->
    I = list_to_integer(atom_to_list(A)),
    F = fib(I),
    io:format("fibonacci ~w = ~w~n", [I, F]),
    init:stop().

fib(0) -> 1;
fib(1) -> 1;
fib(N) ->
    fib(N-2) + fib(N-1).
```

これをシェルでコンパイル、実行してみます。

```
$ erlc fib1.erl
$ erl -noshell -s fib1 main 10 -s init stop
fibonacci 10 = 89
$ erl -noshell -s fib1 main 20 -s init stop
fibonacci 20 = 10946
$ erl -noshell -s fib1 main 30 -s init stop
fibonacci 30 = 1346269
```

16.5 deb パッケージにしてみる

前述のとおり、Erlang は beam ファイルからコードをロードするか、`escript` で実行する必要があります。Erlang のランタイムシステムは、コード自動ロード機構を利用しています。現在のロードパスは Eshell から `code:get_path()` 確認することができます。

```
1> code:get_path().
[".", "/usr/lib/erlang/lib/kernel-2.12.5/ebin",
"/usr/lib/erlang/lib/stdlib-1.15.5/ebin",
"/usr/lib/erlang/lib/xmerl-1.1.10/ebin",
"/usr/lib/erlang/lib/webtool-0.8.3.2/ebin",
"/usr/lib/erlang/lib/typer-0.1.5/ebin",
"/usr/lib/erlang/lib/tv-2.1.4.2/ebin",
"/usr/lib/erlang/lib/tools-2.6.2/ebin",
"/usr/lib/erlang/lib/toolbar-1.3.0.1/ebin",
"/usr/lib/erlang/lib/test_server-3.2.4/ebin",
"/usr/lib/erlang/lib/syntax_tools-1.5.6/ebin",
"/usr/lib/erlang/lib/ssl-3.10/ebin",
"/usr/lib/erlang/lib/ssh-1.0.2/ebin",
"/usr/lib/erlang/lib/snmp-4.12/ebin",
"/usr/lib/erlang/lib/sasl-2.1.5.4/ebin",
"/usr/lib/erlang/lib/runtime_tools-1.7.3/ebin",
"/usr/lib/erlang/lib/public_key-0.1/ebin",
"/usr/lib/erlang/lib/pman-2.6/ebin",
"/usr/lib/erlang/lib/percept-0.7.3/ebin",
"/usr/lib/erlang/lib/parsetools-1.4.5/ebin",
"/usr/lib/erlang/lib/otp_mibs-1.0.4.1/ebin",
"/usr/lib/erlang/lib/os_mon-2.1.8/ebin",
"/usr/lib/erlang/lib/orber-3.6.10/ebin",
"/usr/lib/erlang/lib/odbc-2.10.3/ebin",
"/usr/lib/erlang/lib/observer-0.9.7.4/ebin",
"/usr/lib/erlang/lib/mnesia-4.4.7/ebin",
"/usr/lib/erlang/lib/megaco-3.9.1.1/ebin",
"/usr/lib/erlang/lib/inviso-0.6/ebin",
[...]|...]
```

先頭に、“.”が入っているため、カレントディレクトリにある、beam ファイルは自動的にコードがロードされるため、実行できたわけです。コードパスは /usr/lib/erlang/lib が共通しています。これは、Erlang のデフォルトのライブラリディレクトリで、code:lib_dir() で確認できます。

```
1> code:lib_dir().
"/usr/lib/erlang/lib"
```

先ほど作成した hello モジュールもこのライブラリディレクトリ配下に配置します。しかし、ここに配置すると自動的にロードパスに入るわけではないので、実際に必要な場合は、ちゃんと指定する必要があります。

さて、hello-erlang-1.0 というディレクトリを作り、hello モジュール関係は全てこのディレクトリに移します。

```
$ mkdir hello-erlang-1.0
$ mv hello.erl hello.esc
```

Makefile を用意します。

```
.SUFFIXES: .erl .beam .yrl

.erl.beam:
    erlc -W $<

BINDIR?=/usr/lib/erlang/lib/hello-1.0/ebin

ERL = erl -boot start_clean

# compile erlang module list
MODS = hello

all: compile
    ${ERL} -noshell -s hello start -s init stop

compile: ${MODS:%=%.beam}

install:
    install -d ${DESTDIR}${BINDIR}
    install -m 644 hello.beam ${DESTDIR}${BINDIR}
    install -m 755 hello.esc ${DESTDIR}${usr/bin}/

clean:
    rm -rf *.beam erl_crash.dump
```

dh_make を実行します。今回は CDBS 形式にしてみます。

```
$ dh_make -c gpl -b --createorig
Maintainer name : Kouhei Maeda
Email-Address   : mkouhei@palmtb.net
Date            : Fri, 15 May 2009 00:36:24 +0900
Package Name    : hello-erlang
Version         : 1.0
License         : gpl3
Using dpatch    : no
Using quilt     : no
Type of Package : cdb
Hit <enter> to confirm:
Skipping creating ../hello-erlang_1.0.orig.tar.gz because it already exists
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the hello-erlang Makefiles install into $DESTDIR and not in / .
```

debian ディレクトリ以下を編集します。まず、不要なサンプルファイルを削除します。

```
$ cd debian
$ rm *.ex *.EX
```

changelog を編集します。

```
$ dch
hello-erlang (1.0-1) unstable; urgency=low

 * Initial releas

-- Kouhei Maeda <mkouhei@palmtb.net> Thu, 14 May 2009 22:56:51 +0900
```

control を編集します。erlang-base パッケージが必要なので追記します。

```
Source: hello-erlang
Section: game
Priority: extra
Maintainer: Kouhei Maeda <mkouhei@palmtb.net>
Build-Depends: cdb, debhelper (>= 7), erlang-dev
Standards-Version: 3.8.1
Homepage: http://www.palmtb.net/

Package: hello-erlang
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}, ${erlang-base:Depends}
Description: Hello World for Erlang.
 Hello World Erlang version.
```

copyright を編集します。

```
This package was debianized by:

    Kouhei Maeda <mkouhei@palmtb.net> on Thu, 14 May 2009 22:54:34 +0900

Upstream Author(s):

    Kouhei Maeda <mkouhei@palmtb.net>

Copyright:

    <Copyright (C) 2009 Kouhei Maeda>

License:
(以下略)
```

rules を編集します。dh.make で生成された時は、include 文しかないなので、他を追加します。特に、DEB_FIXPERMS_EXCLUDE で、hello.beam を指定しておかないと、インストール時に実行権限が付与されてしまいます。^{*51}beam ファイルは Erlang VM が読み込むだけですので、実行権は必要ありません。

```
#!/usr/bin/make -f

DEB_MAKE_CHECK_TARGET :=
DEB_FIXPERMS_EXCLUDE := hello.beam

include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/makefile.mk

binary-arch binary-indep: build

DEB_MAKE_INSTALL_TARGET := install DESTDIR=$(DEB_DESTDIR)$(cdb_curpkg)
# Add here any variable or target overrides you need.
```

以上が終わったら、debuild を実行します。

```
$ debuild -us -uc
\end{commandline}

pbuilder を実行します。
\begin{commandline}
$ sudo pbuilder create --distribution sid
$ sudo pbuilder build hello-erlang_1.0-1.dsc
```

問題なければ、最後にインストール、アンインストールできるか確認しておきます。

^{*51} 勉強会当日に分かったことですが、実はインストール先のパスに”bin” という文字列があると実行権限を付与してしまう、CDBS のバグが原因のようです。

```

$ sudo dpkg -i hello-erlang_1.0-1_amd64.deb
未選択パッケージ hello-erlang を選択しています。
(データベースを読み込んでいます ... 現在 193995 個のファイルとディレクトリがインストールされています。)
(hello-erlang_1.0-1_amd64.deb から) hello-erlang を展開しています...
hello-erlang (1.0-1) を設定しています ...

$ dpkg -L hello-erlang
/.
/usr
/usr/bin
/usr/bin/hello.esc
/usr/share
/usr/share/doc
/usr/share/doc/hello-erlang
/usr/share/doc/hello-erlang/changelog.Debian.gz
/usr/share/doc/hello-erlang/README.Debian
/usr/share/doc/hello-erlang/copyright
/usr/sbin
/usr/lib
/usr/lib/erlang
/usr/lib/erlang/lib
/usr/lib/erlang/lib/hello-1.0
/usr/lib/erlang/lib/hello-1.0/ebin
/usr/lib/erlang/lib/hello-1.0/ebin/hello.beam
$ hello.esc
Hello world on Erlang
$ erl -noshell -pa /usr/lib/erlang/lib/hello-1.0/ebin -s hello start -s init stop
Hello world on Erlang
$ sudo apt-get remove --purge hello-erlang
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下のパッケージが自動でインストールされましたが、もう必要とされていません:
(snip)
これらを削除するには 'apt-get autoremove' を利用してください。
以下のパッケージは「削除」されます:
  hello-erlang*
アップグレード: 0 個、新規インストール: 0 個、削除: 1 個、保留: 17 個。
この操作後に 73.7kB のディスク容量が解放されます。
続行しますか [Y/n]?
(データベースを読み込んでいます ... 現在 194002 個のファイルとディレクトリがインストールされています。)
hello-erlang を削除しています ...

```

16.6 まとめ

今回、Erlang の入り口を少しだけ眺めてみました。まだまだ分からないことが多いのでさらに勉強していきます。また、最後に記述したパッケージの作成については、Erlang のコードパスは自動的にロードされないので、パッケージにする = パッケージにされたコードが便利に使える訳ではありません。先月の勉強会で紹介された OCaml では、dh-ocaml というパッケージ作成支援ツールがありますが、Erlang にはありませんでした。Erlang 独自に必要な設定を支援する、dh-erlang のようなツールがあると、Debian でも Erlang はさらに便利になるなと思いました。

16.7 参考

- プログラミング Erlang
ISBN : 978-4-274-06714-3
- Erlang
<http://erlang.org/>
- Erlang - Wikipedia
<http://ja.wikipedia.org/wiki/Erlang>
- Apache CouchDB : The CouchDB Project
<http://couchdb.apache.org/>

17 研究室のソフトウェアを Debian パッケージにしてみる

藤澤 徹



17.1 対象とするソフトウェア

今回パッケージにしてみるソフトウェアは東京大学 近山・田浦研究室で開発されているグリッド用の MPI ライブラリである MC-MPI, および MC-MPI を動作させるのに必要となる, 同じく近山・田浦研究室で開発されている並列分散シェルの GXP の 2 つです.

17.2 パッケージにするに際して

ソフトウェアをパッケージにするには, そのソフトウェアの構成と特徴をよく理解する必要があります. 以下にそれぞれのソフトウェアの特徴のうち, パッケージを作る際に関係のありそうな事を並べます.

17.2.1 MC-MPI

MC-MPI は主に C 言語で記述されたコンパイラとライブラリによって構成されています. また, 一部に Fortran のコードも含まれており, Fortran インターフェースも持っていますが, これはオプションによりオフにする事もできます.

Autotools を使用しており, `./configure && make && make install` という見慣れたコマンドによってビルド, インストールする事ができます.

17.2.2 GXP

GXP は Python で記述されたコマンドと, そのコマンドが必要とする Python モジュールによって構成されています.

インストールする, という作業は想定されておらず, ダウンロードして展開して出てきたディレクトリをどこかに置き, そこにパスを通して使うように作られています.

17.3 使用するツール

Debian のパッケージを作成する方法にはいくつかあるらしいのですが, 今回は debhelper を使用してみます.

17.4 とりあえず動かしてみる

とりあえずは自分の環境で動く事を確かめるため, パッケージとは関係なく使ってみます.

MC-MPI は GXP を必要とするので, まず GXP から試してみましょう.

17.4.1 GXP を試してみる

以下のサイトから執筆時点で最新版である `gxp-3.05.tar.bz2` をダウンロードし、展開します。

<http://www.logos.t.u-tokyo.ac.jp/gxp/>

```
$ mkdir gxp
$ cd gxp
$ # なんとかして gxp-3.05.tar.bz2 を持ってくる
$ tar jxvf gxp-3.05.tar.bz2
$ cd gxp-3.05
```

さて、GXP はインストール不要なので、このままでも動きます。GXP の制御は全て `gxpc` コマンドで行います。

```
$ ./gxpc
gxpc: no daemon found, create one
/tmp/gxp-***-default/gxp-session-***-***-2009-03-20-06-46-07-15360-92311801
```

問題なく動いているようです。

17.4.2 MC-MPI を試してみる

以下のサイトから執筆時点で最新版である `mcmpi-0.21.0.tar.gz` をダウンロードし、展開します。

http://www.logos.ic.i.u-tokyo.ac.jp/~h_saito/mcmpi/

```
$ mkdir mcmpi
$ cd mcmpi
$ # なんとかして mcmpi-0.21.0.tar.gz を持ってくる
$ tar zxvf mcmpi-0.21.0.tar.gz
$ cd mcmpi-0.21.0
```

前述のとおり、Autotools を使用しているの以下の見慣れたコマンドを打ちます。

```
$ ./configure
$ make
$ sudo make install
```

サンプルプログラムが付属しているので、これを `mpicxx` でコンパイルしてみます。

```
$ cd app
$ mpicxx -o hello ./hello.cpp
printf: 28: %q: invalid directive
printf: 28: %q: invalid directive
printf: 28: %q: invalid directive
[ g++ -I/usr/local/include /usr/local/lib/libmpigxp.a -lresolv -lpthread -lnsl -lm ]
/usr/lib/gcc/i486-linux-gnu/4.3.2/../../../../lib/crt1.o: In function '_start':
(.text+0x18): undefined reference to 'main'
collect2: ld はステータス 1 で終了しました
```

おや、動かないようです。調べてみたところどうやら `%q` は使えない事もあるようです。これは文字列を必要ならばクォートするという指定子なので (おそらく)、サクッと `\"%s\"` に置き換えてしまいます。util 以下の `mpicc.in`, `mpicxx.in`, `mpif77.in` に修正を加え、ビルドし直します。

```
$ cd ..
$ make distclean
$ ./configure
$ make
$ sudo make install
```

さて改めてコンパイルします。

```
$ cd app
$ mpicxx -o hello ./hello.cpp
[ g++ -I/usr/local/include "-o" "hello" "./hello.cpp" /usr/local/lib/libmpigxp.a -lresolv -lpthread -lnsl -lm ]
```

今度は上手くいったようです。では実行してみましょう。まず GXP で localhost のみのクラスタを指定し、カレントディレクトリに移動します。そこで mpirun によって実行を開始します。

```
$ gxp
$ gxp use ssh localhost
$ gxp explore localhost
$ gxp cd 'pwd'
$ mpirun -np 1 ./hello
/usr/local/bin/mpirun: 157: Bad substitution
```

おや、またしても失敗です。該当行を見ても

```
done
```

と、よく分からない感じですが、よく調べてみると

```
if [ "${CONF_OPT}" == "" -o "${CONF_OPT:0:1}" == "#" ] ; then
```

の行で落ちています。

`${CONF_OPT:0:1}` という書き方は Bash では動きますが POSIX Shell では動かないようなので、とりあえず Bash で動かすように変更してしまいます。

```
$ cd ..
$ make distclean
$ ./configure
$ make
$ sudo make install
```

では改めて実行しましょう。

```
$ cd app
$ mpirun -np 1 ./hello
INFO: _exchange_end_points: 42498
INFO: _measure_latencies: 23884
INFO: _create_bounding_graph: 26520
INFO: _create_routing_table: 57663
INFO: _create_spanning_tree: 20144
INFO: Env_Init: 172304
Hello 0/1
```

今度こそ見事動きました。

17.5 MC-MPI のパッケージ作成

まずは MC-MPI のパッケージを作成します。

17.5.1 準備

とりあえずさっきとは別のディレクトリで作業をしたほうが良さそうです。

```
$ mkdir deb_mcmpi
$ cd deb_mcmpi
$ # なんとかして mcmpi-0.21.0.tar.gz を持つてくる
$ tar zxvf mcmpi-0.21.0.tar.gz
$ cd mcmpi-0.21.0
```

ここでさっきのバグの修正を加えておきます。

17.5.2 さらに修正

MC-MPI では自身のバージョンを `/usr/etc/VERSION` に記述する事になっているのですが、これはビミョーなので `/usr/share/mcmpi/VERSION` あたりに変更しておきます。変更するファイルは以下のとおりです。

- etc/Makefile.in
- util/mpicc.in
- util/mpicxx.in

- util/mpif77.in
- util/mpirun.in

17.5.3 パッケージ情報記述用のファイルの作成

パッケージを作成する場合にはソフトウェア本体はもちろんの事、インストール方法や依存関係等、そのパッケージの情報を記述したファイルが必要になります。

dh_make コマンドを使用するとこれらを記述するためのファイルの雛形を作成してくれます。この際、DEBFULLNAME, DEBEMAIL 変数により作者情報を指定できます。この名前、メールアドレスが GPG の鍵の情報と一致しないと作成したパッケージにサインできないので困ります。

```
$ export DEBFULLNAME="Tooru Fujisawa"
$ export DEBEMAIL="arai_a@mac.com"
$ dh_make -e arai_a@mac.com -f ../mcmapi-0.21.0.tar.gz
Type of package: single binary, multiple binary, library, kernel module or cdfs?
[s/m/l/k/b]
> s

Maintainer name : Tooru Fujisawa
Email-Address   : arai_a@mac.com
Date            : Fri, 20 Mar 2009 06:55:00 +0900
Package Name    : mcmapi
Version         : 0.21.0
License         : blank
Using dpatch    : no
Type of Package : Multi-Binary
Hit <enter> to confirm:
> [ENTER]
```

途中で Type of package と聞かれます。MC-MPI はライブラリも含まれますが、メインはコンパイラ等なのでとりあえず single binary を選んでおけばよさそうです。

17.5.4 パッケージ情報記述用のファイルの修正

さて、さきほどの dh_make によって、debian というディレクトリが作成され、この中にいろいろなファイルが並んでいます。

この中で重要なのは次のものです。

- changelog
- copyright
- dirs
- control
- rule

順番に修正していきましょう。

changelog

これはパッケージの更新履歴です。ソフトウェア本体の更新履歴とは別モノです。とりあえず雛形に沿って以下のようしておきます。

```
mcmapi (0.21.0-1) unstable; urgency=low

* Initial release

-- Tooru Fujisawa <arai_a@mac.com> Fri, 20 Mar 2009 06:55:00 +0900
```

copyright

これはソフトウェアの著作権情報を記述するファイルです。雛形に沿ってダウンロード元、元々の作者、ライセンス

等を記述します。

```
This package was debianized by Tooru Fujisawa <arai_a@mac.com> on
Fri, 20 Mar 2009 06:55:00 +0900.

It was downloaded from http://www.logos.ic.i.u-tokyo.ac.jp/~h_saito/mcmpi/

Upstream Author(s):
  Hideo Saito <h_saito@logos.ic.i.u-tokyo.ac.jp>

Copyright:
  (c) 2007 Hideo Saito. All Rights Reserved.

License:
  GPL Version 2

The Debian packaging is (C) 2009, Tooru Fujisawa <arai_a@mac.com> and
is licensed under the GPL, see '/usr/share/common-licenses/GPL'.
```

dirs

これはパッケージ作成時に自動的に作成してほしいディレクトリを記述するファイルです。デフォルトでは `usr/bin` と `usr/sbin` が入っていますが、`usr/sbin` は要らないので削除してしまいます。また、`VERSION` を `usr/share/mcmpi` にコピーするようにしたのでこれを追加します。

```
usr/bin
usr/share/mcmpi
```

control

これはパッケージの依存関係や説明を記述するファイルです。

雛形通りに進むと、まず `Homepage` にダウンロード元を書き、`Depends` に次に作成する `gxp` を追加しておきます。また、バグ修正の際に `Bash` を使用する事にしたので `bash` も追加します。最後に説明を短いものと長いものと書いておきます。

```
Source: mcmpi
Subsection: unknown
Priority: extra
Maintainer: Tooru Fujisawa <arai_a@mac.com>
Build-Depends: debhelper (>= 7), autotools-dev
Standards-Version: 3.7.3
Homepage: http://www.logos.ic.i.u-tokyo.ac.jp/~h_saito/mcmpi/

Package: mcmpi
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends} bash gxp
Description: Grid-enabled implementation of MPI
 MC-MPI is a Grid-enabled implementation of MPI, developed by Hideo
 Saito at the University of Tokyo. Its main features include the
 following:
 - [Firewall and NAT traversal]: MC-MPI constructs an overlay
 network, allowing nodes behind firewalls and nodes without global
 IP addresses to participate in computations. There is no need to
 perform manual configuration; MC-MPI automatically probes
 connectivity, selects which connections to establish, and performs
 routing.
 - [Locality-aware connection management]: Establishing too many
 connections, especially wide-area connections, results in many
 problems, including but not limited to the following: exhaustion of
 system resources (e.g., file descriptors, memory), high message
 reception overhead, and congestion between clusters during
 all-to-all communication. Therefore, MC-MPI limits the number of
 connections that are established. If we assume, for simplicity,
 that  $n$  processes are distributed equally among  $c$  clusters, then at
 most  $O(\log n)$  connections are established by each process and at
 most  $O(n \log c)$  connections are established between clusters. As
 MC-MPI uses a lazy connect strategy, fewer connections are
 established for applications in which few process pairs
 communicate. The maximum number of connections allowed can be
 controlled by passing the -beta option to mpirun (see Subsection 3).
 - [Locality-aware rank assignment]: Temporarily disabled in this
 version.
```

rule

これはビルドやパッケージの方法を記述する Makefile です。MC-MPI の場合には Autotools を使っているので得に変える所は無いはずですが。(実はありますがそれは後述...)

17.5.5 ソースパッケージの作成

準備が出来たら `debuild` コマンドでパッケージを作成します。ソースパッケージとバイナリパッケージの両方を作ってみます。

まずはソースパッケージです。

```
$ debuild -S
```

小人さんが頑張ってくれた後、サインをするためのパスフレーズを聞いてくるので2回くらい入力します。するとソースパッケージの完成です。上のディレクトリに色々出来ています。

17.5.6 バイナリパッケージの作成

さて、続いてバイナリパッケージです。

```
$ debuild
```

`configure`, `make` なんか走っている様子が流れていきます。

```
fortran/.libs/libfortran.a(farg.o): In function 'mpigxp_getarg':
*/deb_mcmci/mcmci-0.21.0/src/fortran/farg.f:9: undefined reference to '_gfortran_getarg_i4'
fortran/.libs/libfortran.a(farg.o): In function 'mpigxp_iargc':
*/deb_mcmci/mcmci-0.21.0/src/fortran/farg.f:2: undefined reference to '_gfortran_iargc'
fortran/.libs/libfortran.a(initf.o): In function 'mpi_init__':
*/deb_mcmci/mcmci-0.21.0/src/fortran/initf.c:16: undefined reference to 'mpigxp_iargc__'
*/deb_mcmci/mcmci-0.21.0/src/fortran/initf.c:20: undefined reference to 'mpigxp_getarg__'
collect2: ld returned 1 exit status
```

超怒られました。しかも私の知らない Fortran のコードです。調べてみたところ、この関数は処理系によっては勝手に作られるものだそうで、gFortran は作らないようです。解決する方法が分からないので、ここはいさぎよく Fortran インターフェースを無効にして作り直しましょう。

`debian/rule` ファイルの中で、`./configure` してる行に `--disable-f77` を追加します

```
./configure $(CROSS) --prefix=/usr --mandir=\${prefix}/share/man --infodir=\${prefix}/share/info \
CFLAGS="$CFLAGS" LDFLAGS="-Wl,-z,defs" --disable-f77
```

改めてビルドします。

```
$ debuild
...
/usr/bin/install -c -d /usr/etc
/usr/bin/install: '/usr/etc' の属性を変更できません: No such file or directory
```

またまた怒られました。debuild は `debian/mcmci/` 以下にソフトウェアをインストールしてパッケージにするはずなのに、外にインストールしようとしています。これは `rule` ファイルからディレクトリを `DESTDIR` として渡しているのに、Makefile の方が対応していないためです。

`etc/Makefile.in`, `util/Makefile.in`, の中で `prefix` に `DESTDIR` を追加します。

```
prefix = $(DESTDIR)@prefix@
```

```
$ debuild
```

今度は成功しました。

上のディレクトリに `mcmci_0.21.0-1_i386.deb` が出来ています。

17.5.7 インストールテスト

依存関係が正しいかどうか、インストールしてみましょう。

```
$ cd ..
$ sudo dpkg -i mcmpi_0.21.0-1_i386.deb
未選択パッケージ mcmpi を選択しています。
(データベースを読み込んでいます ... 現在 219582 個のファイルとディレクトリがインストールされています。)
(mcmpi_0.21.0-1_i386.deb から) mcmpi を展開しています...
dpkg: 依存関係の問題により mcmpi の設定ができません:
mcmpi は以下に依存 (depends) します: gxp ... しかし:
  パッケージ gxp はまだインストールされていません。
dpkg: mcmpi の処理中にエラーが発生しました (--install):
依存関係の問題 - 設定を見送ります
以下のパッケージの処理中にエラーが発生しました:
mcmpi
```

gxp が無いと言われました。予定通り作成できているようです。とりあえず削除しておきます。

```
$ sudo apt-get remove --purge mcmpi
```

17.6 GXP のパッケージ作成

さて、無いと言われた GXP パッケージの方を作ります。

17.6.1 準備

こちらもさっきとは別のディレクトリで作業をします。ただし今回、作業を開始した時点では 3.03 が最新バージョンだったので、バージョンアップのテストも兼ねて 3.03 のパッケージをまず作ります。

```
$ mkdir deb_gxp
$ cd deb_gxp
$ # なんとかして gxp-3.03.tar.bz2 を持ってくる
$ tar jxvf gxp-3.03.tar.bz2
$ cd gxp-3.03
```

17.6.2 パッケージ情報記述用のファイルの作成

ほぼ同様です。

```
$ export DEBFULLNAME="Tooru Fujisawa"
$ export DEBEMAIL="arai_a@mac.com"
$ dh_make -e arai_a@mac.com -f ../gxp-3.03.tar.bz2
Type of package: single binary, multiple binary, library, kernel module or cdb?
[s/m/l/k/b]
> s

Maintainer name : Tooru Fujisawa
Email-Address   : arai_a@mac.com
Date            : Fri, 20 Mar 2009 07:15:35 +0900
Package Name    : gxp
Version         : 3.03
License         : blank
Using dpatch    : no
Type of Package : Single
Hit <enter> to confirm:
> [ENTER]
```

GXP は外から見ればコマンド 1 つなので、これも single binary でよさそうです。

17.6.3 パッケージ情報記述用のファイルの修正

さて、これらを記述する前に考えなければならぬことがあります。GXP はインストールを想定されていないため、パッケージにした場合にどこに置いてどのように使うかを決めなければいけません。

今回は /usr/share/gxp 以下にファイルをコピーし、/usr/bin/gxpc を /usr/share/gxp/gxpc にリンクする事にします。

changelog

特に変わった事はしません.

```
gxp (3.03-1) unstable; urgency=low
* Initial release
-- Tooru Fujisawa <arai_a@mac.com> Fri, 20 Mar 2009 07:15:35 +0900
```

copyright

こちらも特に変わった事はしません.

```
This package was debianized by Tooru Fujisawa <arai_a@mac.com> on
Fri, 20 Mar 2009 07:15:35 +0900.

It was downloaded from http://www.logos.t.u-tokyo.ac.jp/gxp/

Upstream Author(s):
Dun Nan
Kenjiro Taura
Yoshikazu Kamoshida

Copyright:
(c) 2008 by Kenjiro Taura. All rights reserved.
(c) 2007 by Kenjiro Taura. All rights reserved.
(c) 2006 by Kenjiro Taura. All rights reserved.
(c) 2005 by Kenjiro Taura. All rights reserved.

License:
  GPL Version 2

The Debian packaging is (C) 2009, Tooru Fujisawa <arai_a@mac.com> and
is licensed under the GPL, see '/usr/share/common-licenses/GPL'.
```

dirs

さて、GXP はインストーラを持っていないので、インストールのコードは直接 rule に書く事になります。できるだけ作業を減らしたいので、必要なディレクトリはこちらに全部書いてしまいましょう。

```
usr/bin
usr/share
usr/share/gxp
```

control

GXP は Python のモジュールを内部に持っているので、これらをインストール先でバイトコードにコンパイルしてあげる作業をしてあげなければいけません。この作業を勝手にしてくれるのが dh_pycentral です。このために、2 箇所 XS-Python-Version: all を追加します。

また、GXP は環境にアーキテクチャに依存しないので、Architecture を all にします。ただし、実行に Python が必要になるので Depends に python を、また前述の作業のために dh_pycentral が必要になるので、Depends, Build-Depends に python-central を追加します。

```
Source: gxp
Subsection: unknown
Priority: extra
Maintainer: Tooru Fujisawa <arai_a@mac.com>
Build-Depends: debhelper (>= 7)
Standards-Version: 3.7.3
XS-Python-Version: all
Homepage: http://www.logos.t.u-tokyo.ac.jp/gxp/

Package: gxp
Architecture: all
Depends: ${shlibs:Depends}, ${misc:Depends}, python
XS-Python-Version: all
Description: parallel/distributed shell
 GXP is a parallel/distributed shell, plus a parallel task execution engine
 that runs your Makefile in parallel on distributed machines.
 Very easy to install
 (no need to compile. install it on YOUR machine and use it on ALL machines).
```

rule

まず, Makefile が無いので \$(MAKE) と入った行は全てコメントアウトします。

そして, \$(MAKE) DESTDIR=\$(CURDIR)/debian/gxp install の行の後に次のようなインストールのコマンドを追加します。展開して出てきたもの全部, としたいのですが, debian ディレクトリがあるのでファイルを並べます。

```
cp -r ChangeLog License README doc ex expectd.py gxpbin gxp gxp.py gxp.py gxp.py gxp.py ifconfig.py inst_local.py \
inst_remote.py inst_remote_stub.py ioman.py misc mkrelease opt.py $(CURDIR)/debian/gxp/usr/share/gxp
ln -s $(CURDIR)/debian/gxp/usr/share/gxp/gxp $(CURDIR)/debian/gxp/usr/bin/gxp
```

また, dh_pycentral を動かすために, dh_python の次の行あたりに dh_pycentral と書いておきます。

```
# dh_python
dh_pycentral
```

17.6.4 ソースパッケージの作成

まずはソースパッケージを作ります。

```
$ debuild -S
```

何事もなく完了します。

17.6.5 バイナリパッケージの作成

さて, 続いてバイナリパッケージです。

```
$ debuild
...
E: gxp: missing-dep-for-interpreter expect => expect (./usr/share/gxp/gxpbin/ssh_passwd)
E: gxp: missing-dep-for-interpreter expect => expect (./usr/share/gxp/gxpbin/su_cmd)
...
```

途中でエラーが出ています。これは ssh_passwd, su_cmd が /usr/bin/expect を使用しているのに, Depends に入っていないという事なので, debian/control を更新します。

```
Depends: ${shlibs:Depends}, ${misc:Depends}, python, python-central, expect
```

ではビルドし直します。

```
$ debuild
```

今度は何事もなく完了し, 上のディレクトリに gxp_3.03-1_all.deb ができています。

17.6.6 バージョンアップ

さて, 作業をしている間に GXP の新しいバージョン 3.05 がリリースされたのでこれをパッケージに反映します。debhelper には新しいバージョンのチェック, 更新を自動化するための機構があります。debian/watch ファイルに

以下のように記述します。(というか, SourceForge なモノは例にあるので拡張子だけ変えます)

```
version=3
http://sf.net/gxp/gxp-(.*)\.tar\.bz2
```

2 行目が最新バージョンの URL のパターンです. ここから自動的に最新バージョンを探して落としてくれます.

```
$ uscan -verbose
-- Scanning for watchfiles in .
-- Found watchfile in ./debian
-- In debian/watch, processing watchfile line:
  http://sf.net/gxp/gxp-(.*)\.tar\.bz2
-- Found the following matching hrefs:
  /sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.02.tar.bz2
  /sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.02.tar.bz2
  /sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.03.tar.bz2
  /sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.03.tar.bz2
  /sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.05.tar.bz2
  /sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.05.tar.bz2
Newest version on remote site is 3.05, local version is 3.03
=> Newer version available from
  http://www.mirrorservice.org/sites/download.sourceforge.net/pub/sourceforge/g/gx/gxp/gxp-3.05.tar.bz2
-- Downloading updated package gxp-3.05.tar.bz2
-- Successfully downloaded updated package gxp-3.05.tar.bz2
  and symlinked gxp_3.05.orig.tar.bz2 to it
-- Scan finished
```

と, いう感じで 3.05 がダウンロードされました. これを展開し, 簡単に現在と同じような構成にすることができます.

```
$ uupdate ../gxp-3.05.tar.bz2
New Release will be 3.05-0ubuntu1.
-- Untarring the new sourcecode archive ../gxp-3.05.tar.bz2
Success! The diffs from version 3.03-1 worked fine.
Remember: Your current directory is the OLD sourcearchive!
Do a "cd ../gxp-3.05" to see the new package
$ cd ../gxp-3.05/
```

はて, 何かバージョンがおかしな事になっています. これは debian/changelog にのみ影響するので, これを書き換えておきます.

```
gxp (3.05-1) unstable; urgency=low

  * New upstream release

-- Tooru Fujisawa <arai_a@mac.com> Fri, 20 Mar 2009 07:32:35 +0900

gxp (3.03-1) unstable; urgency=low

  * Initial release

-- Tooru Fujisawa <arai_a@mac.com> Fri, 20 Mar 2009 07:15:35 +0900
```

あとはビルドし直せば完了です.

```
$ debuild -S
$ debuild
...
E: gxp: ruby-script-but-no-ruby-dep ./usr/share/gxp/gxpbins/tmsub.rb
...
```

さて, 今度は Ruby が必要になったようなので, これを debian/control に追加します.

```
Depends: ${shlibs:Depends}, ${misc:Depends}, python, python-central, expect, ruby1.8
```

ではビルドし直します.

```
$ debuild
```

17.6.7 インストールテスト

GXP の方は特別な依存関係は無いのでインストールできるはず.

```
$ cd ..
$ sudo dpkg -i gxp_3.05-1_all.deb
...
gxp は以下に依存 (depends) します: expect ... しかし:
パッケージ expect はまだインストールされていません。
...
```

と思ったら expect が入ってませんでした。

```
$ apt-get remove --purge gxp
$ sudo apt-get install expect
$ sudo dpkg -i gxp_3.05-1_all.deb
未選択パッケージ gxp を選択しています。
(データベースを読み込んでいます ... 現在 219600 個のファイルとディレクトリがインストールされています。)
(gxp_3.05-1_all.deb から) gxp を展開しています...
gxp (3.05-1) を設定しています ...
```

続いて MC-MPI もインストールします。

```
$ cd ../deb_mcmpi
$ sudo dpkg -i mcmpi_0.21.0-1_i386.deb
未選択パッケージ mcmpi を選択しています。
(データベースを読み込んでいます ... 現在 219699 個のファイルとディレクトリがインストールされています。)
(mcmpi_0.21.0-1_i386.deb から) mcmpi を展開しています...
mcmpi (0.21.0-1) を設定しています ...
```

今度は正しくインストールできました。

18 MC-MPI/GXP 公式パッケージへの道

藤澤 徹



18.1 前回のおさらい

東京大学 近山・田浦研究室で開発されているグリッド用の MPI ライブラリである MC-MPI, および MC-MPI を動作させるのに必要となる, 同じく近山・田浦研究室で開発されている並列分散シェルの GXP の 2 つを Debian パッケージにしてみました.

18.2 今回やること その 1

前回つくったパッケージは割と適当なので, 公式にするには問題が残っています. まずはこれをちよくちよく解決しておきたいと思います.

といっても, とりあえず解決できたのは 1 つだけです.

18.3 MC-MPI で FORTRAN 用ライブラリが無効

これは, 元々のコードが g77 用にかかれていたところを前回出した gFortran でコンパイルしようとしたため, 両者の互換性の無い部分をモロの踏んでしまった事が原因でした.

18.3.1 gfortran_getarg, gfortran_iargc が無い?

```
fortran/.libs/libfortran.a(farg.o): In function 'mpigxp_getarg':  
/home/arai/projects/deb/deb_mcmpi2/mcmpi-0.21.0/src/fortran/farg.f:9:  
undefined reference to '_gfortran_getarg_i4'  
fortran/.libs/libfortran.a(farg.o): In function 'mpigxp_iargc':  
/home/arai/projects/deb/deb_mcmpi2/mcmpi-0.21.0/src/fortran/farg.f:2:  
undefined reference to '_gfortran_iargc'
```

前回の発表では gFortran に getarg, iargc が無い, という風に言ってしまったのですが実際には gFortran に無いわけではなく, gFortran で単体でコンパイルした場合にオブジェクトファイルに含められない事が原因でした.

というわけでライブラリを作る時に libgfortran をリンクする事で解決します.

```
LIBS = @LIBS@ -lgfortran
```

18.3.2 mpigxp_getarg, mpigxp_iargc が無い?

```
fortran/.libs/libfortran.a(initf.o): In function 'mpi_init_':  
/home/arai/projects/deb/deb_mcmpi2/mcmpi-0.21.0/src/fortran/initf.c:16:  
undefined reference to 'mpigxp_iargc_'  
/home/arai/projects/deb/deb_mcmpi2/mcmpi-0.21.0/src/fortran/initf.c:20:  
undefined reference to 'mpigxp_getarg_'
```

これも g77 と gFortran の互換性の問題で、g77 でコンパイルすると元の関数名にアンダースコアが 2 つ追加されるのに対し、gFortran でコンパイルするとアンダースコアが 1 つしか追加されない事が原因です。

というわけでこちらは呼び出し側のコードでアンダースコアの数を調整して解決します。

18.4 今回やること その 2

せっかくパッケージにしたので、これを公式パッケージにして、まっさらな Debian に apt-get でインストールできるようにしてみましょう。

18.5 ところで本当にまだ公式パッケージになってない?

もし他の人の手によって公式パッケージになってたり、途中まで進んでたりしたらそれを知らないまま作業を進めるのは無駄です。

以下のページで作業中のパッケージ等の一覧を見る事ができます。

Work-Needing and Prospective Packages; WNPP

<http://www.debian.org/devel/wnpp/>

とりあえずひととおり検索して、今から作るパッケージがここに無い事を確認しましょう。

18.6 作業開始を宣言するぞ

さきほどのページの下の方に新規項目追加のどうのこうのという節があります。

作業中のものが無いかどうかをこのページで見て確認したわけですから、いざ作業を開始するならばこのページで他の人が見られるようにしなければなりません。

18.7 作業中のパッケージの管理体制

Debian では、作業中のパッケージは Debian バグ追跡システムの上で WNPP という仮想パッケージのバグとして登録され、パッケージ化が完了した時点でこのバグが閉じられます。

つまり、さきほどみたページはこのシステムに登録されている WNPP のバグのリストを表示していた事になります。

18.8 パッケージの登録方法

WNPP にパッケージを登録するには 2 種類の方法が提供されています。

- reportbug を使う
- 自分でメールを出す

18.9 reportbug

reportbug はバグレポートの作成を支援するツールです。

一般的な環境には無いのでインストールしましょう。

```
$ sudo apt-get install reportbug
```

18.10 初期設定

とりあえず起動してみましょう

```
$ reportbug

Welcome to reportbug! Since it looks like this is the first time you have used
reportbug, we are configuring its behavior. These settings will be saved to the
file "/home/arai/.reportbugrc", which you will be free to edit further.
Please choose the default operating mode for reportbug.

1 novice    Offer simple prompts, bypassing technical questions.

2 standard  Offer more extensive prompts, including asking about things that a
            moderately sophisticated user would be expected to know about
            Debian.

3 advanced  Like standard, but assumes you know a bit more about Debian,
            including "incoming".

4 expert    Bypass most handholding measures and preliminary triage routines.
            This mode should not be used by people unfamiliar with Debian's
            policies and operating procedures.

Select mode: [novice]
```

起動すると初期設定の方法を聞かれます。正直に novice と答えましょう。

```
> novice

Please choose the default interface for reportbug.

1 gtk2     A graphical (GTK+) user interface

2 text     A text-oriented console user interface

Select interface:
```

使用するインターフェースは当然 text ですな。

```
> 2

Will reportbug often have direct Internet access? (You should answer yes to
this question unless you know what you are doing and plan to check whether
duplicate reports have been filed via some other channel.) [Y|n|q|?]?

> Y
```

回線が無いと生きていけません。

```
What real name should be used for sending bug reports?
[arai]>

> Tooru Fujisawa

Which of your email addresses should be used when sending bug reports? (Note
that this address will be visible in the bug tracking system, so you may want
to use a webmail address or another address with good spam filtering
capabilities.)
[arai@halko]>

> arai_a@mac.com
```

適当に個人情報を入力します。

```
Do you have a "mail transport agent" (MTA) like Exim, Postfix or SSMTP
configured on this computer to send mail to the Internet? [Y|n|q|?]?
```

このマシンはメールサーバじゃないので n を選択します。

```
> n
Traceback (most recent call last):
  File "/usr/bin/reportbug", line 1841, in <module>
    main()
  File "/usr/bin/reportbug", line 861, in main
    return iface.user_interface()
  File "/usr/bin/reportbug", line 965, in user_interface
    offer_configuration(self.options)
  File "/usr/bin/reportbug", line 496, in offer_configuration
    options=opts, empty_ok=True, force_prompt=True)
TypeError: get_string() got an unexpected keyword argument 'empty_ok'
```

落ちます。

18.11 何が起きたのだろう

...と思ったらアップデートがありました。

```
apt-get install python-reportbug
```

reportbug が 4.2 なのに python-reportbug が 4.0 というよく分からない状態になっていたのが原因のようです。

18.12 初期設定ふたたび

もういちどやりなおしましょう。

```
$ reportbug
...
Do you have a "mail transport agent" (MTA) like Exim, Postfix or SSMTP
configured on this computer to send mail to the Internet? [Y|N|q|?]?
> n
Please enter the name of your SMTP host. Usually it's called something like
"mail.example.org" or "smtp.example.org". If you need to use a different port
than default, use the <host>:<port> alternative format. Just press ENTER if you
don't have one or don't know.
```

MTA の設定は無事通過し、SMTP サーバを聞かれます。reportbug は自動でメールを送信してくれるのですが、今回はこれを使わずにメール送信は自分でやる事にしましょう。

```
> [RET]
Default preferences file written. To reconfigure, re-run reportbug with the "--
configure" option.
```

18.13 パッケージを登録する

さて、初期設定ができたのでパッケージを登録する作業に移ります。

```
Please enter the name of the package in which you have found a problem, or type
'other' to report a more general problem.
```

バグの対象のパッケージを聞かれるので、さきほどの話にあったとおり wnpp を指定します。

```
> wnpp
Are you sure you want to file a WNPP report? [y|N|q|?]?
```

確認されるので Y と入力します。

```

> Y

*** Welcome to reportbug. Use ? for help at prompts. ***
Detected character set: UTF-8
Please change your locale if this is incorrect.

Using 'Tooru Fujisawa <arai_a@mac.com>' as your from address.
Getting status for wnpp...
Will send report to Debian (per lsb_release).
What sort of request is this? (If none of these things mean anything to you, or
you are trying to report a bug in an existing package, please press Enter to
exit reportbug.)

1 ITP This is an 'Intent To Package'. Please submit a package description
along with copyright and URL in such a report.
2 O The package has been 'Orphaned'. It needs a new maintainer as soon as
possible.
3 RFA This is a 'Request for Adoption'. Due to lack of time, resources,
interest or something similar, the current maintainer is asking for
someone else to maintain this package. They will maintain it in the
meantime, but perhaps not in the best possible way. In short: the
package needs a new maintainer.
4 RFH This is a 'Request For Help'. The current maintainer wants to continue
to maintain this package, but they needs some help to do this, because
their time is limited or the package is quite big and needs several
maintainers.
5 RFP This is a 'Request For Package'. You have found an interesting piece of
software and would like someone else to maintain it for Debian. Please
submit a package description along with copyright and URL in such a
report.

Choose the request type:

```

まずバグの種類を聞かれます。wnpp に登録するバグには上記のように 5 種類あります。簡単に説明すると以下のとおりです。

ITP これからあるソフトウェアのパッケージを作成するぞ、という宣言
O 自分のパッケージを手放すという宣言
RFA 自分のパッケージの里親になってくれという要求
RFH 誰か手伝ってくれという要求
RFP 誰かパッケージにしてくれという要求

というわけで ITP を選びます

```

> 1
Please enter the proposed package name:

```

パッケージ名を要求されますまずは mcmpi のパッケージを作る事にしましょう。

```

> mcmpi
Checking status database...
Please briefly describe this package; this should be an appropriate short
description for the eventual package:

```

説明文はパッケージを作った時に決めたのでそのままです。

```

> Grid-enabled implementation of MPI

Your report will be carbon-copied to debian-devel, per Debian policy.
Querying Debian BTS for reports on wnpp (source)...
3082 bug reports found:

Outstanding bugs -- Important bugs; Unclassified (1 bug)
  1) #502809 0: potracegui -- KDE frontend for potrace, severity it normal
(1-1/3082) Is the bug you found listed above [y|N|m|r|q|s|f|?]?

```

登録済みの 3082 個のバグに重複が無いかどうかチェックしろという大変素晴らしい要請を受けました。回避する策を探しましょう。

```
> ?
y - Problem already reported; optionally add extra information.
N - (default) Problem not listed above; possibly check more (skip to Next
page).
m - Get more information about a bug (you can also enter a number without
selecting "m" first).
r - Redisplay the last bugs shown.
q - I'm bored; quit please.
s - Skip remaining problems; file a new report immediately.
f - Filter bug list using a pattern.
? - Display this help.
(1-1/3082) Is the bug you found listed above [y|N|m|r|q|s|f|?]?
```

ほらあるじゃないですか。

```
> s
Spawning emacs...
```

ほら回避できた。まあさきほどウェブ上でもチェックしたし、とりあえずは良しとしましょう。

さて、Emacs が起動されてバグレポートの雛形が表示されました。バージョン、上流作者、URL、ライセンス、使用言語、長い説明を書きましょう。どれもパッケージを作る時にもう書いたものなのでコピペで OK です。

```
Subject: ITP: mcmpi -- Grid-enabled implementation of MPI
Package: wnpp
Owner: Tooru Fujisawa <arai_a@mac.com>
Severity: wishlist

*** Please type your report below this line ***

* Package name      : mcmpi
  Version           : 0.21.0
  Upstream Author   : Hideo Saito <h_saito@logos.ic.i.u-tokyo.ac.jp>
* URL               : http://www.logos.ic.i.u-tokyo.ac.jp/~h_saito/mcmpi/
* License           : GPL
  Programming Lang: C, FORTRAN
  Description       : Grid-enabled implementation of MPI

MC-MPI is a Grid-enabled implementation of MPI, developed by Hideo
Saito at the University of Tokyo. Its main features include the
following:
- [Firewall and NAT traversal]: MC-MPI constructs an overlay
network, allowing nodes behind firewalls and nodes without global
IP addresses to participate in computations. There is no need to
perform manual configuration; MC-MPI automatically probes
connectivity, selects which connections to establish, and performs
routing.
- [Locality-aware connection management]: Establishing too many
connections, especially wide-area connections, results in many
problems, including but not limited to the following: exhaustion of
system resources (e.g., file descriptors, memory), high message
reception overhead, and congestion between clusters during
all-to-all communication. Therefore, MC-MPI limits the number of
connections that are established. If we assume, for simplicity,
that n processes are distributed equally among c clusters, then at
most O(log n) connections are established by each process and at
most O(n log c) connections are established between clusters. As
MC-MPI uses a lazy connect strategy, fewer connections are
established for applications in which few process pairs
communicate. The maximum number of connections allowed can be
controlled by passing the -beta option to mpirun (see Subsection 3).
- [Locality-aware rank assignment]: Temporarily disabled in this
version.
```

こんな感じで保存しましょう。

```
Report will be sent to "Debian Bug Tracking System" <submit@bugs.debian.org>
Submit this report on wnpp (e to edit) [Y|n|a|c|e|i|l|m|p|q|?]?
```

メールサーバは持っていないので Y は選べません。

```
> ?
Y - (default) Submit the bug report via email.
n - Don't submit the bug report; instead, save it in a temporary file (exits
  reportbug).
a - Attach a file.
c - Change editor and re-edit.
e - Re-edit the bug report.
i - Include a text file.
l - Pipe the message through the pager.
m - Choose a mailer to edit the report.
p - Print message to stdout.
q - Save it in a temporary file and quit.
? - Display this help.
Submit this report on wnpp (e to edit) [Y|n|a|c|e|i|l|m|p|q|?]?
```

ファイルに保存してくれれば自分で送れるので, q を選びましょう.

```
> q
reportbug stopped; your incomplete report is stored as "/tmp/reportbug-
wnpp-20090516-28182-HoQ43f". This file may be located in a temporary directory;
if so, it might disappear without any further notice. To recover this file to
use it as bug report body, please take a look at the "-i FILE, --include=FILE"
option.
```

というわけでさっきのバグレポートを submit@bugs.debian.org に送って完了です.

reportbug を使わなくても自分で全部書くという手もあります.

メールを送るとバグに ID が振られ, そのスレッドのトップとしてさっきのメールが返ってきます.

しばらくするとウェブ上の WNPP から参照可能になります.

さて, 表明もしてしまったところで, これからパッケージの手直しを頑張ろうと決意して今回はおしまいです.



19 Debian GNU/Linux 上で Android SDK を使ってみた

上川純一

19.1 はじめに

Android は Open Handset Alliance の提供する携帯プラットフォームです。携帯電話ではめずらしくオープンソースの集大成であるということで、Debian 開発者としては試してみずにはいられません。Debian だからどうということはありませんが、とりあえず、SDK 1.5r1 がリリースされたのを機にそろそろ試してみることにします。

19.2 利用機材

- Android 開発用の携帯電話, Android 1.5 を実行
- Debian GNU/Linux lenny + squeeze on MacBook

19.3 ダウンロード・インストール

Debian 上で開発するために、各種必要なパッケージなどをインストールします。

19.3.1 必要な Debian パッケージのインストール

Java や 32bit エミュレーション環境のパッケージなどをまずインストールしておきます。

```
# apt-get install ia32-libs sun-java6-bin sun-java6-jdk
```

x86_64 の 64bit 環境で 32bit のアプリケーションを利用できるように、ia32-libs をインストールします。また、java もインストールしておきます。

19.3.2 SDK ダウンロード

Android の SDK をダウンロードしてきます。Debian 用は Linux を選択します。i386 版しかないのですが、注意。amd64 でも i386 エミュレーション上で稼働するので利用できます。

http://developer.android.com/sdk/1.5_r1/index.html

```
$ unzip android-sdk-linux_x86-1.5_r1.zip
$ ls -l android-sdk-linux_x86-1.5_r1
合計 24
-rw-rw---- 1 dancer dancer 193 2009-04-22 13:15 RELEASE_NOTES.html
drwxrwx--- 3 dancer dancer 4096 2009-04-23 10:50 add-ons
drwxrwx--- 12 dancer dancer 4096 2009-05-01 16:13 docs
-rw-rw---- 1 dancer dancer 176 2009-04-22 13:15 documentation.html
drwxrwx--- 4 dancer dancer 4096 2009-04-23 10:50 platforms
drwxrwx--- 3 dancer dancer 4096 2009-04-22 13:14 tools
```

tools ディレクトリにパスを通しておきます。 .bashrc 辺りに下記を記述しておきます。

```
export PATH="${PATH}":path/to/android/android-sdk-linux_x86-1.5_r1/tools/
```

19.3.3 eclipse ダウンロード

Debian パッケージになっているバージョンは古いため、eclipse のサイトからダウンロードしてきます。

<http://www.eclipse.org/> でバージョン Ganymede の x86-64 アーキテクチャ向けのバイナリをダウンロードしてきました。

展開すればそこで ./eclipse を実行すればエディタが起動します。

```
$ tar xzf eclipse-java-ganymede-SR1-linux-gtk-x86_64.tar.gz
$ ls eclipse
about.html      configuration  eclipse.ini    libcairo-swt.so  plugins
about_files     dropins       epl-v10.html  notice.html      readme
artifacts.xml  eclipse       features      p2
$ cd eclipse/
$ ./eclipse
```

19.3.4 Eclipse に Android 用プラグインの追加

Eclipse のプラグインをインストールします。 http://developer.android.com/sdk/1.5_r1/installing.html を参考に設定します。うまく動かない部分はメーリングリスト検索を駆使し、正しい方法を探します。

プラグインは <http://dl-ssl.google.com/android/eclipse/site.xml> から Android DDMS と Android Development Tools をインストールすればよいようです。

あと、設定する内容としては、Window の Properties の Android でさきほど展開した SDK のパスを指定します。

19.3.5 Android デバイス接続

udev の設定の修正が必要です。 /etc/udev/rules.d/z60_android.rules に下記記述します。

```
SUBSYSTEM=="usb_device",
SYSFS{idVendor}=="0bb4",MODE="0666",
SYMLINK+="k"
```

接続できていれば、adb コマンドで接続を確認できます。

```
$ adb devices
List of devices attached
HT-----   device
```

adb shell コマンドでシェルが起動します。

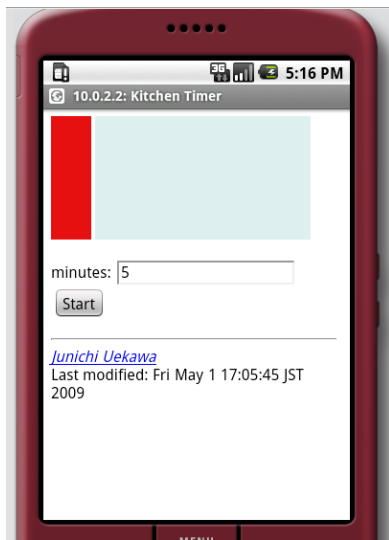
```
$ adb shell
```

19.4 emulator を試してみる

まず、コマンドラインから emulator を起動するのを試してみました。emulator の起動には AVD (Android Virtual Device) の作成がまず必要なようです。とりあえず適当に作成してみました。

```
$ android list targets
[選択可能なターゲットの一覧]
$ android create avd -n uekawadroid -t 2
Android 1.5 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
Created AVD 'uekawadroid' based on Android 1.5
$ emulator -avd uekawadroid
```

エミュレータは Android の実機をあたかうのと同じような感覚で操作できます。Android のブラウザを起動したりできます。エミュレータ内部から見ると、10.0.2.2 がホスト OS のアドレスになります。HTML+javascript で作成したキッチンタイマーを動作させて見たところ、動作しました。



19.5 Android の簡単なアプリケーションを作成して起動してみる

それでは、Android 用の簡単なアプリケーションを書いてみましょう。

<http://developer.android.com/guide/developing/eclipse-adt.html> にしたがって淡々と作成します。まず、File の New Project で Android のをプロジェクト選択して作成しました。ターゲットビルドは 1.1 にとりあえずしると書いてありますが、よくわからないので、手元のファームウェアのバージョン 1.5 にあわせておきました。

Run の Run(Ctrl-F11) で Android Application を選択するとエミュレータを起動してアプリケーションを実行することができます。最初に生成されたソースコードのままの状態だと Hello World アプリケーションが起動します。

19.6 Android の簡単なアプリケーションを書いてみる

アプリケーションの作成やエディタの起動の部分はなんとかなるようになったので、次はチュートリアルを眺めてみます。Hello World あたりがよさげです。

<http://developer.android.com/guide/tutorials/hello-world.html>

一通り読んで、とりあえずサンプルを眺めてどうやれば文字列を表示できるのかを確認したので、好きな文字列を表示するアプリケーションを作ってみます。

Android の top コマンドでプロセスの一覧を出力できるので、コマンドを実行してその出力を表示するだけのアプリケーションを作成してみました。中核をなす `src/jp.gr.netfort.dancer/TopView.java` は以下になりました。

```

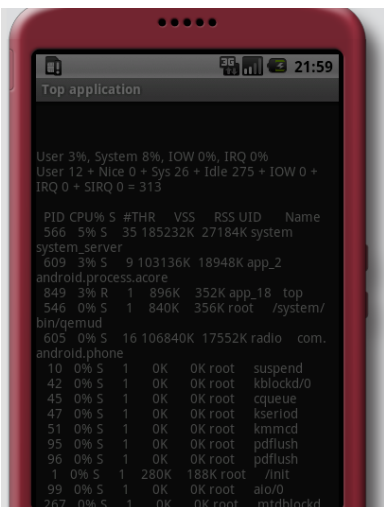
package jp.gr.netfort.dancer;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import java.io.*;

public class TopView extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        String [] command = { "top", "-n", "1"};
        String output = "";

        Runtime runtime = Runtime.getRuntime();
        Process process = null;
        try {
            process = runtime.exec(command);
        } catch (Exception exception){
            System.exit(1);
        }
        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
        String line;
        try {
            while((line = reader.readLine()) != null) {
                output = output + line + "\n";
            }
        } catch (Exception exception) {
            System.exit(1);
        } finally {
            try {
                reader.close();
            } catch (Exception exception) {
                System.exit(1);
            }
        }
        TextView tv = new TextView(this);
        tv.setText(output);
        setContentView(tv);
    }
}

```



19.7 Android の簡単なアプリケーションを実機で稼働させてみる

実際に使う便利なアプリケーションを作ってみましょう。ライトニングトークを実施する際にどのプレゼンテーションが気に入ったかを表明する、投票用に利用するボタンでも作成しましょうか。

19.7.1 エフェクト用のデータ

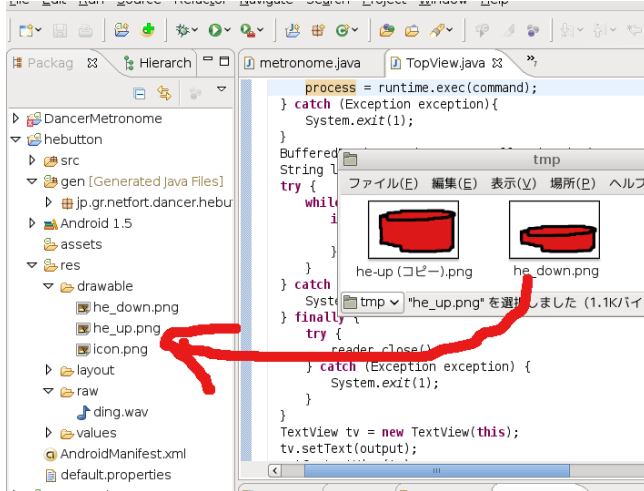
まず、なんらかの効果音が必要です。以前作成した ding.wav をひろってきます。res/raw 以下に nautilus からドラッグアンドドロップでコピーします。

あと、ボタン風味の画像が必要です。適当に gimp で絵をかいて png ファイルを作成しました。

nautilus から res/drawable 以下に png ファイルをドラッグアンドドロップでコピーします。

ファイル名からそのままリソースの名前を作成するため、-は利用できないようです、_ は利用できるようです。最

初 he-up.png を作成しようとして、エラーが出たので he_up.png に変更しました。



19.7.2 レイアウトの作成

おもむろに ImageButton を作成して、src に画像リソースを追加しておきましょう。

19.7.3 クリックアクションの作成

R.id.ImageButton01 に対して onClickListener を作成し、クリックされた瞬間の反応を作成しましょう。

画像をクリックされた状態に切り替えるには ImageButton の setImageResource() を呼び出せばよいようです。

音を出すには、MediaPlayer のインスタンスを作成し、start() を呼び出せばよいようです。

一定時間経過してから元の画像に戻すには、アラームを設定してコールバックしてもらえるようにして、コールバックを呼び出されたときに処理をするようにすればよいようです。

19.7.4 サーバ側のアクション

それでは、クリックされたときにどういふことを行ふのかのアクションを作成してみましょう。HTTP サーバを準備します。こういうイロモノ系のサーバを作成するときには upaccho2 サーバを使うことが上川家の常識なので、それを踏襲します。(予定)

19.8 おまけ:開発環境に必要なメモリ消費は節減できるのか?

手元の MacBook には 1GB しかメモリを搭載していないです。eclipse はデフォルトの設定だと起動した瞬間に 1GB 近くの仮想メモリを消費し、即 Swap を使用してしまうという問題がありました。

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
17124 dancer 20 0 972m 302m 20m S 1 31.2 0:39.85 java
```

とりあえず、ヒントを求めて maps あたりを眺めてみます。それなりにたくさん mmap していますが、それだけでは説明できないようなメモリ領域を確保しています。

```
$ cat /proc/17124/status | grep Vm
VmPeak:      1056748 kB
VmSize:      1003316 kB
VmLck:        0 kB
VmHWM:       310740 kB
VmRSS:       221616 kB
VmData:      788504 kB
VmStk:        84 kB
VmExe:        36 kB
VmLib:       38812 kB
VmPTE:       1108 kB
$ cat /proc/17124/maps | grep rw | sort -rn -k5
```

とりあえず Java のアプリケーションの Heap の使い方がよくわからないので、jconsole で接続して問題を解析し

ます。

メモリプールがいくつかあることがわかります。

- Tenured Gen が 60MB
- Survivor 1MB
- Code Cache が 4MB
- Perm Gen が 70-90MB
- Heap は 100MB(GC したら 50MB 程度)

eclipse.ini が現状こうなっているのですが、これをおそろおそろチューニングしてみます。

```
-showsplash
org.eclipse.platform
-framework
plugins/org.eclipse.osgi_3.4.2.R34x_v20080826-1230.jar
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Xms40m
-Xmx256m
-XX:MaxPermSize=256m
```

起動時に確保したヒープメモリの領域を開放していない部分については、GC を頻繁に行えば起動は若干遅くなりますがうまくいけるような気がします。Xmx 128m に変更してみると、若干メモリの消費が下がりました。しかしそこまで劇的ではありませんね。MaxPermSize を 64MB にするとエラー

```
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: PermGen space
```

が発生しました。96MB だとエラーでおちないようです。

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	コメント
17124	dancer	20	0	972m	302m	20m	S	1	31.2	0:39.85	java
19663	dancer	20	0	795m	296m	23m	S	1	30.6	0:36.69	Heap 128MB 版
20266	dancer	20	0	661m	292m	22m	S	1	30.1	0:41.62	MaxPermSize 96MB

がんばってみた割にはメモリが 10MB 程度しか節約できませんでした。エミュレータも起動していると 500MB くらい swap にいってしまうので、つらいです。しかも、ギリギリのメモリ設定にしていたところ、いろいろと操作しているとメモリが足りないというエラーで落ちました。こりゃダメだ。メモリ買いにいくかな。

```
-showsplash
org.eclipse.platform
-framework
plugins/org.eclipse.osgi_3.4.2.R34x_v20080826-1230.jar
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Xms40m
-Xmx128m
-XX:MaxPermSize=96m
```

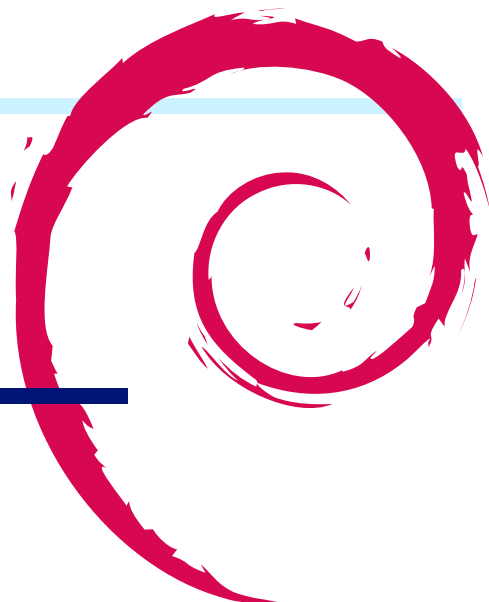
19.9 参考文献

本記事の作成に参考にした文献です。

- <http://www.eclipse.org/>: eclipse
- <http://developer.android.com/>: Android のページ、開発情報がまとまっている。特に http://developer.android.com/sdk/1.5_r1/index.html: Android SDK 1.5 のダウンロードページからたどるページが重要。
- <http://developer.android.com/guide/basics/what-is-android.html>: SDK の開発マニュアル。SDK の `./android-sdk-linux_x86-1.5_r1/docs/` に同じものがあるのでオフラインでも安心。ただしオンラインのドキュメントは微妙な訂正がアップデートされ続けているようなのでネットワーク接続が利用できるオンラインの場合はそちらを参照するほうがよいかもしれません。

20 DDTSS 活用

上川 純一



20.1 はじめに

最近 apt の国際化も進み、Debian archive の Description の翻訳もパッケージの配布の仕組みと一緒に配布されるようになってきました。一方、翻訳を用意するプロジェクト (DDTP:Debian Description Translation Project) はまだ普及していません。今回はその活用を事前課題として、今後どうやってうまく活用していくのかを考えます。

20.2 Debian のパッケージ説明文の翻訳の目的とは

Debian Distribution には数万のパッケージがあります。パッケージの説明文はインストールするソフトウェアを探すのに寄与する内容である必要があります。

現在、その説明文はすべて英語で提供されています。しかし、英語で記述されているとそもそも英語を理解していないとわからない部分があり、しかも技術的な専門用語をまじえた説明文章は日本語を主言語とするユーザにとって理解しにくい場合があります。

そもそもの Description が翻訳に適するレベルに至っていない、日本語に訳しにくい文章であれば、もとの文書を訂正するように依頼しましょう。

また、翻訳した Description が日本語として理解しにくい、技術的な表現が不適切な表現であれば、その Description は適切ではないでしょう。

また、Debian 全体で表記をできるだけ統一しましょう。同じ内容を説明するのに表記のゆれがあるとパッケージの説明文をみているときにその差異が目がいってしまいます。

その他に考慮する点は?

20.3 DDTP

DDTP の説明は Debian.org のページにあります: <http://www.debian.org/intl/l10n/ddtp.ja.html>。 <http://www.debian.or.jp/community/translate/description-ja.html> などとあわせて眺めてください。

メールフロントエンドや、ウェブフロントエンドなどがあります。

20.4 DDTSS とは

DDTP のフロントエンドの一つが DDTSS です。 <http://ddtp.debian.net/ddtss/index.cgi/xx> に提供されています。

20.5 DDTSS 活用の手順

20.5.1 アカウントの登録

ログイン画面^{*52}に飛ぶと、アカウントを作成するためのリンク (Create Login^{*53}) があるので、そこからアカウント登録します。

メールが届くので、そこからリンクをクリックしてアカウントをアクティベートすると、ログイン画面からログインできるようになります。

20.5.2 パッケージ説明文の翻訳作成

<http://ddtp.debian.net/ddtss/index.cgi/ja> 画面で、Pending Translation の欄にあるパッケージを適切にクリックすると、パッケージ説明文の編集画面が登場します。一部の文書が翻訳されておらず<trans>となっていたり、まったく翻訳されていなかったり、状態はまちまちです。

この画面で翻訳を作成します。

20.5.3 レビューの実施

<http://ddtp.debian.net/ddtss/index.cgi/ja> 画面で、Pending Review の欄にあるパッケージを適切にクリックすると、パッケージ説明文のレビュー画面が登場します。

レビューして変な部分があれば訂正しましょう。

20.5.4 debian-doc メーリングリストでのレビュー実施

debian-doc というメーリングリストで議論とかレビューはおこなうみたいです。

20.6 参考文献

以下のページも参考にしてください。

- Debian パッケージの説明文を日本語で読みたい! ~DDTP へのお誘い~<http://www.debian.or.jp/community/translate/description-ja.html>
- 武藤健志さんの blog の『Debian ドキュメント翻訳手続き』: <http://kmuto.jp/d/index.cgi/debian/debian-doc-procedure.htm>
- 小林儀匡さんの Debian 勉強会 2006 年 9 月資料「翻訳への誘い」: tokyodebian.alioth.debian.org/pdf/debianmeetingresume200609.pdf
- debian-doc メーリングリスト: 主要な議論が行われています。質問なども、こちらで。

^{*52} <https://ddtp.debian.net/ddtss/index.cgi/login>

^{*53} <https://ddtp.debian.net/ddtss/index.cgi/createlogin>

21 パッケージの依存性を解く：EDOS から Mancoosi まで

Ralf Treinen



注) Ralf Treinen さんの発表用スライドを、簡単かつ適当に抄訳しました。訳の間違いなど、苦情は倉敷まで。

21.1 Mancoosi プロジェクト

- ヨーロッパにおける研究プロジェクト
- 期間：2008/2 - 2011/1
- EDOS プロジェクト (2004/1 - 2007/6) の後継

21.2 EDOS と Mancoosi の目的

- コンピュータサイエンスの研究成果をフリー/オープンソースソフトウェア (F/OSS) ディストリビューションに応用：
 - 形式手法：論理に基づくツールと手法 from Verification
 - ツール from ソフトウェア工学
- F/OSS ディストリビューションと、科学研究において、到達水準を向上させるのが目的

21.3 なぜ科学研究者が F/OSS に興味をもつのか？

F/OSS インフラストラクチャが特に興味深いのは：

- 中心となる設計者がいない
- 開発が速く、分散されている
- パッケージの相互依存が強い
- 巨大なコードベース
- 同時に複数の CPU アーキテクチャにパッケージを提供
- 複数の OS に対して同時にパッケージを提供

21.4 なぜ F/OSS が科学の研究成果に興味をもつのか？

F/OSS における品質保証の現状は：

- コンパイルとインストールの (時には自動化された) テスト

- テストケースの自動生成はなされない
- 自動検証ツールは使われていない
- 個別のパッケージと、ディストリビューション全体について、品質保証のための自動化されたツールが必要に迫られている

21.5 パッケージの依存関係における EDOS workpackage

焦点：リリース管理者の観点によるディストリビューションの一貫性

質問：規定のリポジトリにあるパッケージしか使えない場合に、ユーザが選択したパッケージはインストール可能だろうか？

リリース管理者や品質保証チームが、問題のある、もしくは無用なパッケージを見つける助けになる質問

21.6 パッケージのインストーラビリティ as SAT (命題論理の充足可能性判定問題)

パッケージの関係について、命題論理に基いて数学的なモデルを作ることから始めた。

- 各パッケージ p (バージョンは v) は、ブール変数 P_v (P_v であればパッケージはインストール可能。そうでなければ不可能) として解釈される
- 依存関係は、それぞれ連鎖として解釈される。例： $aterm \quad libc6 \wedge (libice6 \vee xlibs) \wedge \dots$
- パッケージ a と b での衝突は、公式 $\neg(a \wedge b)$ として解釈される

定理：

あるパッケージ p のバージョン v は、 P_v を真とする真偽値が存在し、リポジトリの記号化を満足すれば、「インストール可能」である

21.7 パッケージインストールの容易さ as SAT - 例

平均的な式は 400 の定数を持つ。KDE のインストールでは 32000

21.8 EDOS ツールチェイン

EDOS を通じていくつかのツールが開発された (OCaml で 110000 行のコード)

edos-debcheck

パッケージのインストーラビリティをチェックするコマンドラインツール

pkglab

インタラクティブなコンソール環境で、リポジトリの検査用

ceve

パッケージリストの複数フォーマットをパース/変換する

tart

リポジトリを分割 (メディアなど) する。この時、 i 番目のメディアに含まれているパッケージは、 i 番目までのメディアを使えばインストール可能となる

いくつかを詳細に見てみよう.....

21.9 edos-debcheck

- edos-debcheck は APT パッケージリスト ($/var/lib/apt/lists/*$ など) をもとに、ある/いくつかの/全てのパッケージがインストール可能かどうかをチェックする

カスタマイズされた SAT 解決器がベースになっていて、とても高速：testing/amd64 の main にある全パッケージのインストーラビリティのチェックにかかる時間は、エントリーレベルのマシン上で 5 秒。

大きな成功例

emdebian ではパッケージのアップロード前に edos-debcheck を使って、壊れたパッケージをアーカイブにアップロードしないようにしている

Debian のパッケージ名：edos-debcheck, edos-rpmcheck

21.10 pkglab

- pkglab はインタラクティブな、コンソールベースの環境で、パッケージベースのソフトウェアディストリビューションのリポジトリを調査するもの

機能：

- 現在と過去のパッケージリストを読み込む
- パッケージのインストーラビリティをチェック (edos-debcheck で)
- 関数型のクエリ言語 (map, filter, fold, ...)
- リポジトリがどのような変遷を経て発展してきているかを調べる (edos-debcheck 単体では不可能)

Debian のパッケージ名：dose2 (基本のライブラリ), ceve (パッケージリストのパーサ/コンバータ), pkglab (インタラクティブ環境), edos-debcheck (debian 向けのインストーラビリティチェッカ), edos-rpmcheck (RPM 向けのインストーラビリティチェッカ)

21.11 Debian で、インストールできないパッケージを見つける

Debian, Skilelinux, Debian GNU/kFreeBSD において、毎日 edos-debcheck を使ってインストール不可能なパッケージをモニターしている：

<http://edos.debian.net/edos-debcheck>

インストールできないパッケージの頻出ケース：

1. autobuilder の追従 (例えば、arch:all のパッケージが arch:any のパッケージと同時にアップロードされ、autobuilder が遅延している)：通常の一過性のインストール不可状態
2. a が b に依存しており、全アーキテクチャに b がない：b のビルドに問題があったり、a において、とっておおらかなアーキテクチャ指定がなされている (厳密にすべき)
 - 上記の特殊な例：a は arch:all。最近の Install-To フィールドを追加する提案はこれに対処するもの (#436733)
3. 深刻なパッケージのバグで、実際に修正する必要がある :-)

21.12 Debian で、宣言されていない衝突を見つける

ユーザにこういったエラーを見せる前に取り除く。

1. 愚直に：パッケージ全部 (200,000,000) を同時にインストールしてみる.....そんなバカな!
2. 少なくとも 1 ファイルを共有しているペアのみ考慮 (Contents を使えば簡単)：867 ペア (2008/4/16 の amd64/sid)
3. 依存性のうえで同時インストール可能なペアに絞る (pkglab を使えば簡単)：102 ペア
4. diversion によって誤検知されているかも：ペアでのインストールを chroot 環境でテスト：27 のバグもちパッケージペアが検出された

レポート : <http://edos.debian.net/missing-conflicts/>

BTS:ユーザ treinen@debian.org、タグ `edos-file-overwrite`

21.13 Mancoosi プロジェクト

アップグレード問題 = インストールされたパッケージのローカルなステータスを変更するようメタインストーラが要求することで発生する問題。

アップグレード問題の解決は、いくつかの理由で失敗し得る :

- 呼び出しエラー、依存性解決、パッケージ受信、パッケージ展開、メンテナスクリプトの実行、.....

Mancoosi は 2 つの側面からアップグレード問題に取り組もうとしている :

ロールバックのサポート

想定外のエラー (メンテナスクリプトなど) があった場合、事後のリカバリが唯一の解決策

依存性解決

メタインストーラが最新の水準を満たしていない (例えば、不完全性 : when there is one、解決法を見つけることができない) : 我々がうまくやらねば !

21.14 依存性の解決に必要なもの

完全性

アップグレード問題への解決法が存在するなら、メタインストーラはそれを発見できなくてはならない

最適性

同等の異なる複数の解決法を区別するため、最適化の基準を指定できる必要がある。例えば :

- ダウンロードサイズを最小化
- 使用するディスクスペースを最小化
- J. Random DD (DD の誰か) がメンテしているパッケージのブラックリスト
- などなど

効率

依存性の解決は、可能な限り高速でなくてはならない

21.15 依存性解決法の competition

我々は、依存性解決における魔法の銀の弾丸となるアルゴリズムを探そうとしているわけではもちろんないが、依存性解決の competition 運営につながる助けとなることができる

- popcon のように収集した現実におけるアップグレード問題 (メタインストーラが持つオプトインのデータ収集プラグイン)
- 様々な証跡 : 平易な分析 (スピード)、最適化した分析 (よりよい解法)、.....
- 開発者と研究者が、独自アルゴリズムの独自実装を提示することができる
- 勝者は幸運と栄光をゲット

SAT 解決自体のような関連分野において、同様の competition が技術水準を最新に押しあげるのに大いに寄与しています。なぜこの分野でやらないのでしょうか ?

22 Debian 勉強会の資料を作成しよう

山下 尊也



22.1 リポジトリのチェックアウト

```
$ git clone git://git.debian.org/git/tokyodebian/monthly-report
```

```
$ make -j4 # エラーがでます  
$ cp -p git-pre-commit.sh .git/hooks/pre-commit  
$ make -j4
```

22.2 emacs whizzytex の準備

```
$ cd XXXXX  
$ emacs
```

whizzytex^{*54}を起動します。プレビューが開始するというメリットだけでなく、即時コンパイルエラーを検出できるのが重要です。

```
M-x whizzytex-mode
```

スペルチェックも起動しましょう。^{*55}

```
M-x flyspell-mode
```

22.3 自分のセクションを追加する

まず自分のセクションを追加します。

```
\dancersection{セクション名}{名前}  
\index{XXXX@YYYY}  
\label{XXXX@YYYY}  
  
% 本文
```

^{*54} メンテナの上川さんによる資料が <http://www.netfort.gr.jp/~dancer/column/whizzytex.html.ja> にあります。

^{*55} iamerican もしくは ibritish パッケージが必要です。

22.4 セクションを追加

22.3 でセクションを追加しましたので、次にサブセクションを追加していきます。必要があれば、サブサブセクションと追加していきます。

```
\subsection{xxxx} % サブセクション (小節)
\subsection{xxxx}
\subsubsection{xxxx} % サブサブセクション (少々節)
```

22.5 注釈を追加する

現時点で CDBS を使用しているソースパッケージは 1805 個、ソースパッケージ全体の ~ 16% だそうです*⁵⁶。

```
現時点で CDBS を使用しているソースパッケージは 1805 個、
ソースパッケージ全体の  $\sim 16\%$  だそうです
\footnote{これ、正しいですかね? }。
```

22.6 画像を追加する

画像を追加する方法はいくつか方法がありますが、eps ファイルだとファイルサイズが大きくなるため、png や jpg などに変換を行なった上で追加した方が良いと思います。

```
M-!
Shell command [~/monthly-report]$ cd image200808; ebb colinux_xlaunch_xdmcp.png
```

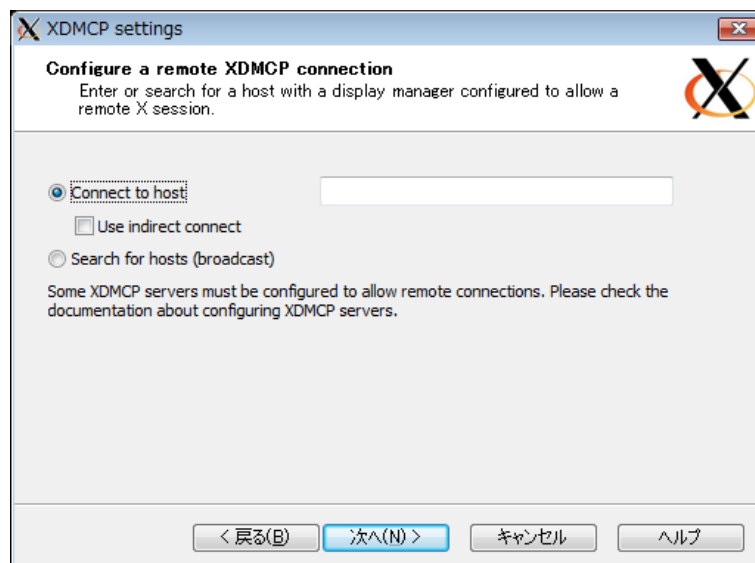


図 5 XLaunch 起動

*⁵⁶ これ、正しいですかね?

```

\subsection{画像を追加する}
\label{sec:addpicture}

\begin{figure}[htbp]
\begin{center}
\includegraphics[width=100mm]{image200808/colinux_xlaunch_xdmcp.png}
\end{center}
\caption{XLaunch 起動}
\label{fig:colinux_xlaunch_xdmcp}
\end{figure}

```

22.7 図や表を参照する

monthlyreport では、図や表を参照するのに便利な `\fgrep{}` と `\tbref{}` が定義されています。

22.6 で紹介した図 5

```
\ref{sec:addpicture}で紹介した\fgrep{fig:colinux_xlaunch_xdmcp}
```

22.8 URL を追加してみる

関西 Debian 勉強会の詳細については、<http://wiki.debian.org/KansaiDebianMeeting> にあります。

```
関西 Debian 勉強会の詳細については、\url{http://wiki.debian.org/KansaiDebianMeeting}にあります。
```

22.9 表を作ってみる

\LaTeX で表を作る事も出来ますが、私は OpenOffice Calc で表を作成し、 \TeX のコードを生成してくれる Calc2Latex^{*57} を使っています。

22.10 箇条書き

来年度の予定は、

- Live Helper のハンズオン
- 翻訳ハンズオン
- ...

Debian のインストールは、

1. 言語設定
2. キーボードの選択
- ...

Emacs が好きな理由

- ♡ カスタマイズ可能
- ♡ 指が覚えている
- ♡ Emacs の中だけでほとんどの作業が出来る
- ...

^{*57} <http://calc2latex.sourceforge.net/indexj.html>

```

\begin{itemize}
  \item Live Helper のハンズオン
  \item 翻訳ハンズオン\
\end{itemize}

\begin{enumerate}
  \item 言語設定
  \item キーボードの選択\
\end{enumerate}

\begin{list}%
  {\heartsuit} %default label
  {Emacs が好きな理由} %formatting parameter
  \item カスタマイズ可能
  \item 指が覚えている
  \item Emacs の中だけでほとんどの作業ができる\
\end{list}

```

22.11 その他の注意事項

22.11.1 改行

LaTeX では、一般的な改行の解釈とは異なります。空の行で段落の区切りと判断します。段落として判断したら、行の頭に 1 文字分全角の空白が入ります。

22.11.2 コマンドの結果について

コマンドの結果などを簡単に載せられるように kansaimonthlyreport.sty(元は、monthlyreport.sty) では、commandline という勉強会用に定義されており、commandline を利用した方が簡単です。

22.11.3 エスケープが必要な文字列の出力方法

- \ verb は見にくくなるので最終的手段
- ~
- ^
- aaaa<aaaa そのままだと:;j
- aaaa>aaaa そのままだと:;j
- aaaa#aaaa
- aaaa%aaaa
- _aaaa
- /usr/share/iso-codes/iso_3166.tab
- /usr/share/iso-codes/iso_3166.tab

```

\begin{itemize}
  \item {\tt /usr/share/iso-codes/iso\_3166.tab}
  \item /usr/share/iso-codes/iso\_{}3166.tab
\end{itemize}

```

22.11.4 全角英字などの禁止

一般的に全角英字、半角カナの使用は使用しません。これらの背景については、検索してみてください。

23 LaTeX beamer でプレゼンテーション

大浦 真



23.1 はじめに

- Debian 勉強会などでは、講師をする際にプレゼン用の資料が必要になる。
- どうやって作るか。
 - MS Power Point や Openoffice.org の Impress。
 - Magic Point。
 - HTML とブラウザ。
 - テキストファイルと less。
 - \LaTeX で作る。

23.2 LaTeX beamer

23.2.1 特徴

- \LaTeX がベース。
 - 特別なことはしていないので、 \LaTeX さえ分かれば使える。
- 最新版は 2007 年 3 月リリースの 3.07。
- 豊富なテーマ、豊富なサンプルが用意されている。
- 簡単なアニメーションくらいはできる。
- `\section`、`\subsection` を普通に使える。
- 開発が盛ん。
 - マニュアルは全 224 ページ。

23.2.2 インストール

- L^AT_EX 一式が必要。Debian なら texlive をインストールする。
- latex-beamer のインストール
 - <http://latex-beamer.sourceforge.net/> からダウンロード。
 - latex-beamer 以外にも pgf と xcolor も必要。
 - Debian だったら、# apt-get install latex-beamer
 - 手でインストールする場合は、/usr/share/texmf などに texmf というディレクトリがあるので、texmf/tex/latex/beamer などのサブディレクトリを作ってそこにスタイルファイルをコピー。

23.2.3 基本的な使い方

- クラスファイルとして beamer.cls を使う。

```
\documentclass{beamer}
```

- 使うテーマを決める。

```
\usetheme{Boadilla}
```

- タイトル、作者、日付の設定。

```
\title{LaTeX beamer でプレゼンテーション}  
\author{大浦@ Debian.org}  
\date{2008 年 12 月 14 日 (日)}
```

- スライド一枚一枚はフレームと呼ばれる。フレームは `\begin{frame}` と `\end{frame}` で囲む。
- タイトルページを作る。

```
\begin{frame}  
\titlepage{  
\end{frame}
```

- 必要なら目次のページを作る。
 - 本文中で `\section{}` などを使えば自動的に目次が作られる。テーマにもよるが、スライドの中にセクション名が表示される。

```
\begin{frame}  
\frametitle{目次}  
\tableofcontents{  
\end{frame}
```

- 適宜セクションに分ける。
- プレゼンテーションは複数のフレームでできている。
- `\frametitle{}` がそのページのタイトル。

```
\begin{frame}  
\frametitle{はじめに}  
\begin{itemize}  
\item Debian 勉強会などでは、講師をする際にプレゼン用の資料が必要になる。  
.....  
\end{itemize}  
\end{frame}
```

23.2.4 フレームとスライド

- フレームは複数のスライドからできている。
- `\pause{}` を使うと一つのフレームを複数に分割できる。

```

\begin{frame}
\frametitle{はじめに}
\begin{itemize}
\item Debian 勉強会などでは、講師をする際にプレゼン用の資料が必要になる。
\pause{}
\item どうやって作るか。
.....
\end{itemize}
\end{frame}

```

- 特定のスライドだけでコマンドを有効にできる。
- \LaTeX のコマンドの後に $\langle \rangle$ を付ける。

```

\begin{itemize}
\item \textbf{この行は常に太字です。}
\item \textbf<2>{この行は 2 枚目のスライドでのみ太字です。}
\item \textbf<3>{この行は 3 枚目のスライドでのみ太字です。}
\end{itemize}

```

23.2.5 簡単なアニメーション

- ごく簡単なアニメーションが使える。
- `\animate` と `\animatevalue` を使う。

```

\newcount\opaqueness
\begin{frame}
.....
\animatevalue<1-10>{\opaqueness}{100}{0}
\begin{colormixin}
{\the\opaqueness!averagebackgroundcolor}
この文字列はフェードアウトします。
\end{colormixin}
\end{frame}

```

23.2.6 画像の取り込み

- \LaTeX なので普通に画像も取り込める。
- 例えば Debian の logo, `openlogo.eps`
- `graphicx.sty` を使う。

```

\includegraphics[height=3cm,keepaspectratio,clip]{openlogo.eps}

```

23.3 プレゼンの方法

- dvi ファイルを作って、xdvi でプレビューすると画像が表示されないが、作成途中のプレビューには十分。
- プレゼンの際には、Postscript か PDF に変換する。
- Postscript にする場合は、dvips を使う。
 - `beamer.cls` にオプション `dvips` を付ける。 `\documentclass[dvips]{beamer}`
 - `pspresent` を使って表示。 <http://www.cse.unsw.edu.au/~matthewc/pspresent/>
- PDF にする場合は、dvipdfmx か、Postscript に変換してから Ghostscript (`ps2pdf`) を使う。
 - `dvipdfmx` を使うためには、`beamer.cls` にオプション `dvipdfm` を付ける。
 - Acrobat Reader を使って表示。 `hyperlink` の機能も使える。
- 配布用の資料ではアニメーションなどは無効にする。アニメーションは複数ページとして扱われるので、それを 1 ページにする。
 - `beamer.cls` にオプション `handout` を付ける。
- Postscript から印刷する場合は、`psutils` に含まれる `psnup` で 1 ページに複数のフレームを入れて印刷。
 - `psutils` <http://www.dcs.ed.ac.uk/home/ajcd/psutils/>

23.3.1 終わりに

- L^AT_EX の使い方さえ分かっていたら latex-beamer は簡単に使えます。
- L^AT_EX 上の他のプレゼンテーションスタイルよりも派手なスライドを作れます。
- ぜひ、latex-beamer を使って Debian 勉強会で講師をしましょう。



24 各種イベント開催実績

上川 純一

Debian 開発者の育成を目的として開催してきた東京エリア Debian 勉強会も今回で 4 年目が終了しました。事前課題事後課題を設定しており、予習復習を必要だとうたっている勉強会ですが、実際にどれくらい実施されているのか、まずは事前課題の実施実績を確認してみましょう。図 6 です。各月の数字は表 3 です。

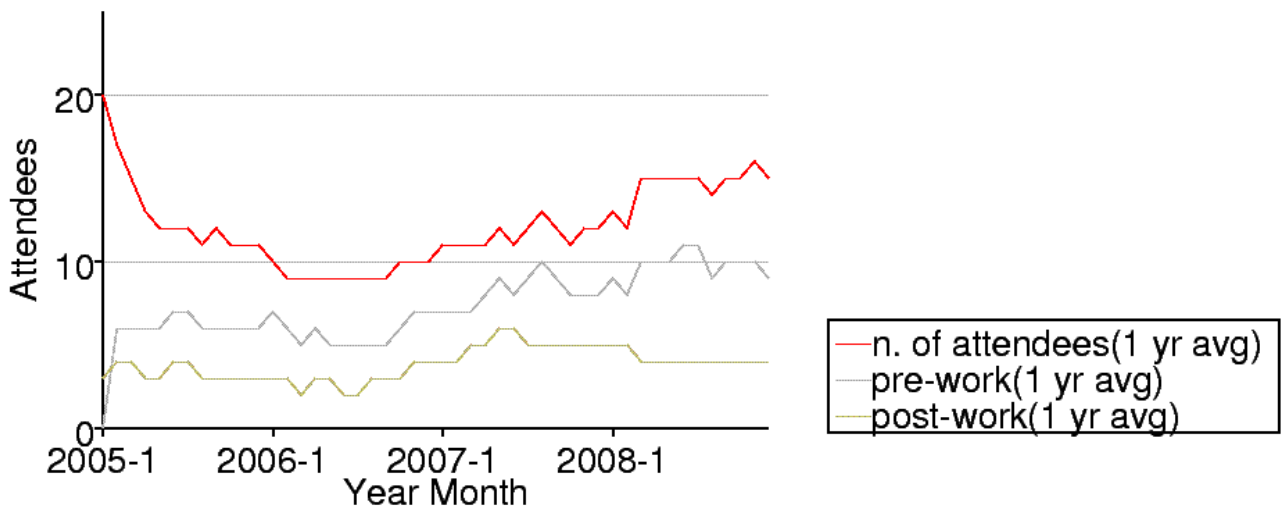


図 6 東京エリア Debian 勉強会事前課題・事後課題提出実績 (12ヶ月移動平均)

昨年からは勉強会は東京と関西の両方で開催されるようになっていきます。東京エリア Debian 勉強会と、関西 Debian 勉強会について簡単に出席数で実績をみてみましょう。まず、東京についてグラフにしてみると図 7 になります。

具体的な数字と、トピックを見てみましょう。

それでは、関西 Debian 勉強会の出席状況を確認してみましょう。グラフで見ると図 8 になります。表で見ると表 8 になります。

表3 東京エリア Debian 勉強会事前課題・事後課題提出実績

年	月	参加人数	事前課題	事後課題
2005	1	20	0	3
2005	2	15	12	6
2005	3	10	8	4
2005	4	9	6	2
2005	5	6	6	4
2005	6	13	10	5
2005	7	12	7	4
2005	8	9	6	2
2005	9	14	7	4
2005	10	10	5	3
2005	11	7	6	3
2005	12	8	5	3
2006	1	9	7	3
2006	2	8	4	2
2006	3	6	0	0
2006	4	15	11	6
2006	5	7	2	1
2006	6	14	9	4
2006	7	2	2	4
2006	8	17	9	7
2006	9	12	8	5
2006	10	22	15	7
2006	11	3	12	7
2006	12	15	7	4

年	月	参加人数	事前課題	事後課題
2007	1	15	6	4
2007	2	13	8	4
2007	3	0	6	16
2007	4	18	14	6
2007	5	21	14	7
2007	6	1	0	1
2007	7	18	12	3
2007	8	25	18	5
2007	9	0	7	5
2007	10	10	1	6
2007	11	19	10	6
2007	12	11	11	4
2008	1	22	11	4
2008	2	0	1	0
2008	3	37	27	11
2008	4	17	13	3
2008	5	20	14	3
2008	6	10	8	2
2008	7	17	12	4
2008	8	10	0	4
2008	9	17	13	5
2008	10	11	0	7
2008	11	22	14	6

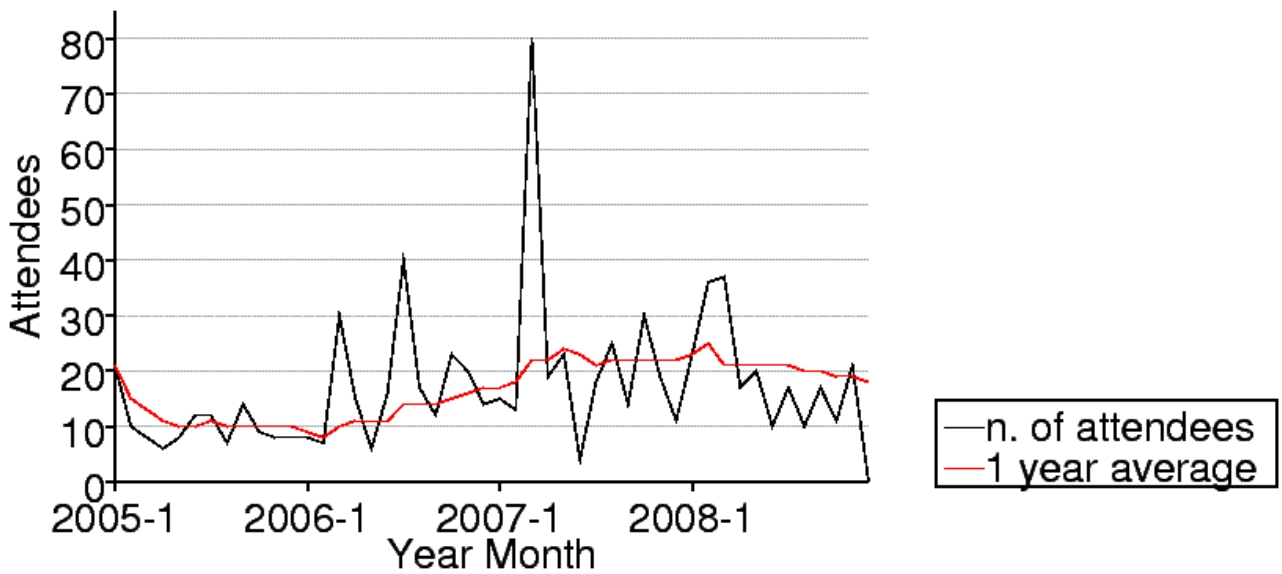
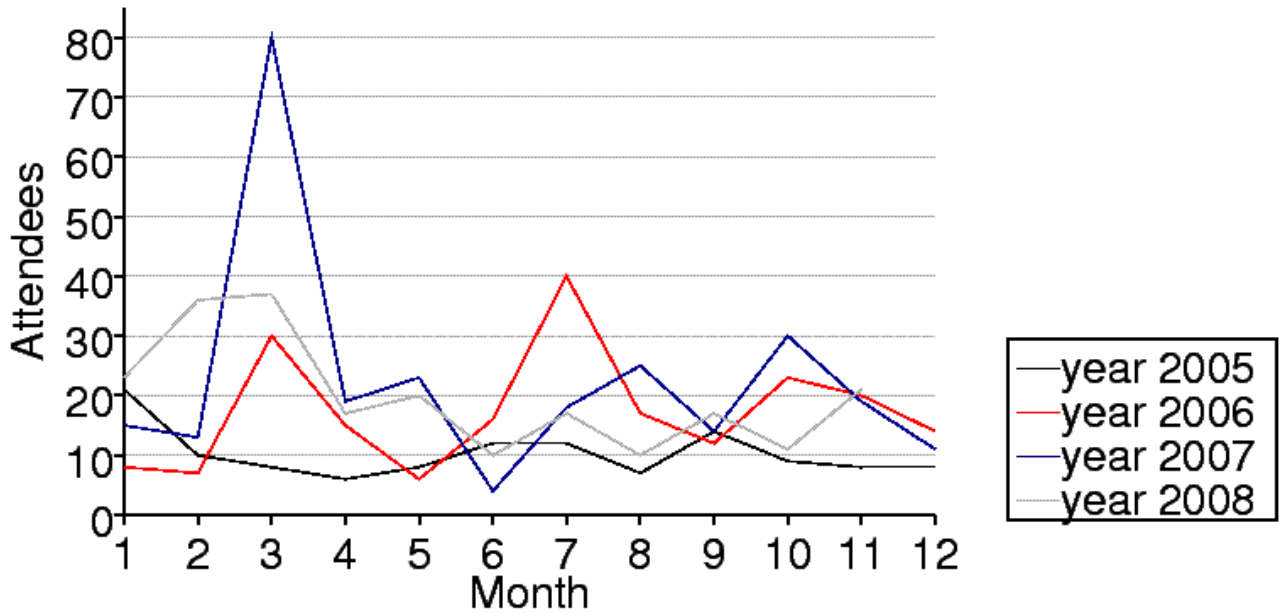


図7 東京エリア Debian 勉強会の参加人数推移

表 4 東京エリア Debian 勉強会参加人数 (2005 年)

	人数	内容
2005 年 1 月	21	秘密
2005 年 2 月	10	debhelper1
2005 年 3 月	8	(早朝) debhelper2、social contract
2005 年 4 月	6	debhelper3
2005 年 5 月	8	DFSG、dpkg-cross、lintian/linda
2005 年 6 月	12	alternatives、d-i
2005 年 7 月	12	toolchain、dpatch
2005 年 8 月	7	Debconf 参加報告、ITP からアップロードまで
2005 年 9 月	14	debconf
2005 年 10 月	9	apt-listbugs、バグレポート、debconf 翻訳、debbugs
2005 年 11 月	8	DWN 翻訳フロー、statoverride
2005 年 12 月	8	忘年会

表 5 東京エリア Debian 勉強会参加人数 (2006 年)

	参加人数	内容
2006 年 1 月	8	policy、Debian 勉強会でやりたいこと
2006 年 2 月	7	policy、multimedia
2006 年 3 月	30	OSC: debian 勉強会、sid
2006 年 4 月	15	policy、L ^A T _E X
2006 年 5 月	6	mexico
2006 年 6 月	16	debconf、cowdancer
2006 年 7 月	40	OSC-Do: MacBook Debian
2006 年 8 月	17	13 執念
2006 年 9 月	12	翻訳、Debian-specific、oprofile
2006 年 10 月	23	network、i18n 会議、Flash、apt
2006 年 11 月	20	関西開催: bug、sid、packaging
2006 年 12 月	14	忘年会

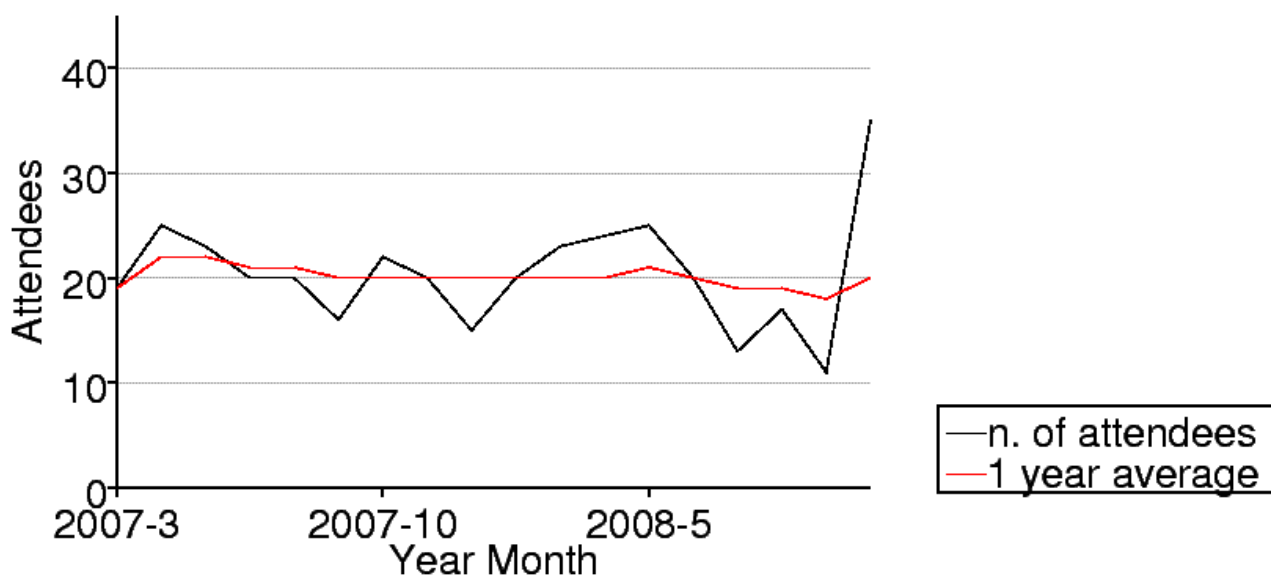


図 8 関西の参加人数推移

表7 東京エリア Debian 勉強会参加人数 (2008 年)

	参加人数	内容
2008 年 1 月	23	一年を企画する
2008 年 2 月 29+1 日	36	OSC
2008 年 3 月	37	データだけのパッケージ、ライセンス
2008 年 4 月	17	バイナリパッケージ
2008 年 5 月	20	複数のバイナリパッケージ
2008 年 6 月	10	debhelper
2008 年 7 月	17	Linux kernel patch / module パッケージ
2008 年 8 月	10	Debconf IRC 会議と Debian 温泉
2008 年 9 月	17	po4a, 「Debian メンテナのお仕事」
2008 年 10 月	11?	OSC Tokyo/Fall
2008 年 11 月	17	「その場で勉強会資料を作成しちゃえ」 Debian を使った L ^A T _E X 原稿作成合宿
2008 年 12 月	?	忘年会

表6 東京エリア Debian 勉強会参加人数 (2007 年)

	参加人数	内容
2007 年 1 月	15	一年を企画する
2007 年 2 月	13	dbcs, dpatch
2007 年 3 月	80	OSC 仮想化
2007 年 4 月	19	quilt, darcs, git
2007 年 5 月	23	etch, pbuilder, superh
2007 年 6 月	4	エンジンパラ開催 : Debconf7 実況中継
2007 年 7 月	18	Debconf7 参加報告
2007 年 8 月	25	cdn.debian.or.jp
2007 年 9 月	14	exim
2007 年 10 月	30	OSC Tokyo/Fall(CUPS)
2007 年 11 月	19	live-helper, tomooyo linux kernel patch, server
2007 年 12 月	11	忘年会

表8 関西 Debian 勉強会参加人数 (2007 年)

	参加人数	内容
2007 年 3 月	19	開催にあたり
2007 年 4 月	25	goodbye、youtube、プロジェクトトラッカー
2007 年 6 月	23	社会契約、テーマ、debian/rules、bugreport
2007 年 7 月	20 前後	OSC-Kansai
2007 年 8 月	20	Inkscape、patch、dpatch
2007 年 9 月	16	ライブラリ、翻訳、debtorrent
2007 年 10 月	22	日本語入力、SPAM フィルタ
2007 年 11 月	20 前後	KOF
2007 年 12 月	15	忘年会、iPod touch

表9 関西 Debian 勉強会参加人数 (2008 年)

	参加人数	内容
2008 年 2 月	20	PC Cluster, GIS, T _E X
2008 年 3 月	23	bug report, developer corner, GPG
2008 年 4 月	24	coLinux, Debian GNU/kFreeBSD, sid
2008 年 5 月	25	ipv6, emacs, us-tream.tv
2008 年 6 月	20	pbuilder, hotplug, ssl
2008 年 8 月	13	coLinux
2008 年 9 月	17	debian mentors, ubiquity, DFSG
2008 年 10 月	11	cdbs,cdn.debian.or.jp
2008 年 11 月	35	KOF
2008 年 12 月	?	



25 2008 年を振り返ってみる

上川 純一

25.1 最近のトレンドと今後の推移

最近どんなことがあって、これからどういうことがあるでしょうか。みんなで予想してみましょう。

2006	2007	2008	2009	2010
IntelMac に、core-duo で dual-core CPU に、glantank(ARM)、OpenMicroServer(MIPS)、OpenSolaris が出て Debian/Solaris (Nexenta) 登場、SparcT1 がオープンに、CC3.0、Qwik 登場 (?), 雑誌が大多数消失、	VT・AMD-V(仮想化技術) が普及 (ML115!), 黒箱 (ARM)、OpenBlocks(PPC?), iPhone 登場、HSDPA 月額 5000 円くらいに、google mobile、VISTA リリース、Leopard リリース、GPL3.0、メモリ 2G がコモディティに、SparcT2 がオープン、ニコニコ動画、	python 3.0 ruby 1.9 wine 1.0, wine64 登場 RoR 2.0 登場で普及に 4 コア・64bit の CPU がデスクトップに普及、Core2Quad 値下げ。ニコニコ動画 1000 万ユーザ突破、初音ミクブームに地デジ関連の PC 製品の普及勉強会の普及 (楽天とか) 公衆無線 LAN (wireless gate) 携帯電話の売上が落ちる、iPhone, Android 登場、emobile 100 円 PC 抱きあわせ (eeePC, Dell mini9) Zaurus 販売終了。Chumby 発売。サーバの仮想化 ESXi・シンククライアント MacBook Air 発売、無線 802.11n が実機に SystemZ10 発表 世界経済の崩壊 (IT 投資緊縮 財政、職を失う人が増加) FreeBSD 7 (malloc, ZFS ?) Debian 次世代育成計画始動 Debian Maintainer 制度始動 セキュリティー関連 (OpenSSL 事件、DNS 事件) クラウド関連が流行? Nintendo DSi	tile window manager boom ? Lenny リリース予定 Debian 合同結婚式 デスクトップ、8 コア、4GB? 8GB? ノートパソコン、2 コア、2GB? Linux が標準インストールの PC。SSD の値段と容量がこなれる? HDD がなくなる?高くなる? ファイルシステムかわる? ipv6 使えるようになってる? Bluray が普及? DL 禁止法? torrent に逆風?	消費税上昇に伴う繁忙期 クラウドにより、単純なホスティング業者がつかない? 一部は自社でもつようになる? Squeeze リリース USB 3.0 搭載、wireless USB vs Bluetooth ? 組み込み CPU は Atom に統一?Arm は残ってる? kFreeBSD オフィシャルアーキテクチャに ruby 2.0 リリース?

25.2 SWOT

できたこと	できなかったこと	チャンスとなるもの	脅威となるもの
<p>嫁ができた。(20%、10% 2次元) 将来のDDが生まれた。</p> <p>Hands-on(パッケージ作成と$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$) と合宿(温泉とプレスト)、DMCができた。</p> <p>持ち回りで発表する場所がいろいろあった。(上川不在) 他の勉強会(カーネル読書会)に殴り込み(岩松、山根)</p> <p>Ubuntuとの交流、Debian JPに寄付</p>	<p>当初の目標であった女子高生、大学生への勧誘が成功しなかった。</p> <p>嫁ができなかった(70%、想像力がたりない)</p> <p>発表やりたかったけどできなかった(でんさん)</p> <p>GNU/Hurd、SuperH</p> <p>Debconfにいけなかった(上川以外)</p> <p>日本へのDebconf招致活動未完</p>	<p>所帯持ちハック方法が生まれる(いかにして時間をつくるか)。</p> <p>無駄な買い物をしなくなる。ハックしないと。</p> <p>東京オリンピックの成立?</p> <p>学生の就職率低下、大学院生増加? ニート増加?</p> <p>GPLv3の普及?</p> <p>Android?</p>	<p>環境整備ができなくなる(ボーナスが減った、仕事なくなるかも)</p> <p>AtomによるCPUアーキテクチャーの駆逐</p> <p>ハックできないデバイスの増加(電話とか)</p> <p>法制度の強化により自由がうばわれる?</p> <p>従量制課金に移行?</p>

25.3 SWOT 2

		チャンスとなるもの	脅威となるもの
		<p>所帯持ちハック方法が生まれる (いかにして時間をつくるか)。 無駄な買い物をしなくなる。ハックしないと。 東京オリンピックの成立? 学生の就職率低下、大学院生増加?ニート増加? GPLv3 の普及? Android?</p>	<p>環境整備ができなくなる (ボーナスが減った、仕事なくなるかも) Atom による CPU アーキテクチャーの駆逐 ハックできないデバイスの増加 (電話とか) 法制度の強化により自由がうばわれる? 従量制課金に移行?</p>
できたこと	<p>嫁ができた。(20%、10% 2次元) 将来の DD が生まれた。 Hands-on(パッケージ作成と L^AT_EX) と合宿 (温泉とブレスト)、DMC ができた。 持ち回りで発表する場所がいろいろだった。(上川不在) 他の勉強会 (カーネル読書会) に殴り込み (岩松、山根) Ubuntu との交流、Debian JP に寄付</p>	<p>東京オリンピックの会場でハック。 学校にお願いして学生を勧誘する・ハンズオン。 「君のさわっている XX は Linux だけどより詳しく知りませんか?」(ミニノートのプリインストール、携帯、Android)</p>	<p>よりよいネットワークプロトコルを実施 アンチ Atom?</p>
できなかったこと	<p>当初の目標であった女子高生、大学生への勧誘が成功しなかった。 嫁ができなかった (70%、想像力がたりない) 発表やりたかったけどできなかった (でんさん) GNU/Hurd、SuperH Debconf にいけなかった (上川以外) 日本への Debconf 招致活動未完</p>	<p>Debconf にいく。 時間をつくる、ライフハック (所帯持ちハック)。 専門学校、工業高校、大学での開催。 言語系のコミュニティーに切り込む インフラ系の人たちに切り込む 非常勤講師になる。</p>	

26 Debian Trivia Quiz

上川 純一



ところで、みなさん Debian 関連の話題においついていますか？ Debian 関連の話題はメーリングリストをよんでいると追跡できます。ただよんでいるだけでははりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

26.1 第 50 回勉強会

第 50 回勉強会の出題範囲は `debian-devel-announce@lists.debian.org` に投稿された内容と Debian Project News からです。

問題 1. 2 月 14 日にリリースされたのは

- A etch
- B sarge
- C lenny

問題 2. 3 月 12 日にリリースされた Debian Policy のバージョンは

- A 3.8.0
- B 3.8.1
- C 3.8.2

問題 3. Debian Data Export は何か

- A Debian パッケージに関するデータを容易に取り寄せるためのインターフェイス
- B 全世界にある、あらゆるデータを Debian パッケージ化するプロジェクト
- C Debian のあらゆるデータを Google にすべて喰わせてみましたというプロジェクト

問題 4. Olivier Berger が ML に流した `bts-link` に関する情報は

- A `bts-link` のソース管理リポジトリが壊れました
- B `bts-link` でリンクできる BTS を増やしました
- C `bts-link` サービスは終了です

問題 5. Debian FTP master が変わりました。誰が新しく入ったでしょうか。

- A Kei Hibino
- B Ryan Murray
- C Mark Hymers

問題 6. 3月16日に Joerg Jaspert が ML に流したアナウンスは

- A Debian の新しいロゴが決まりました
- B パッケージのセクションが追加されます
- C いくつかのディストリビューションと吸収合併します

問題 7. 3月21日にリリースされた pbuilder のバージョンは?

- A 0.187
- B 1.298
- C 3.1

問題 8. Debian.org DPL に立候補していないのは

- A Stefano Zacchiroli
- B Steve McIntyre
- C Nobuhiro Iwamatsu

問題 9. Debian JP 選挙の立候補しめきりは?

- A 3月20日
- B 3月22日
- C 3月24日

26.2 第51回勉強会

第51回勉強会の出題範囲は `debian-devel-announce@lists.debian.org` に投稿された内容と Debian Project News からです。

問題 10. 4月9日にアップデートされた etch のバージョンは?

- A 4.0r8
- B 4.0beta8
- C etch-a-sketch

問題 11. 4月11日にアップデートされた lenny のバージョンは?

- A 5.0r1
- B 5.0.1
- C lenny++

問題 12. Debian.org DPL になったのは?

- A Stefano Zacchiroli
- B Steve McIntyre
- C Nobuhiro Iwamatsu

問題 13. Debian JP Leader になったのは?

- A Kei Hibino
- B Hiroyuki Yamamoto
- C Nobuhiro Iwamatsu

問題 14. Debian で新しく追加されたアーキテクチャは？

- A GNU/kFreeBSD i386/amd64
- B GNU/kMinix-3.0 i386/amd64
- C GNU/kOpenDarwin i386/amd64

26.3 第 52 回勉強会

第 52 回勉強会の出題範囲は `debian-devel-announce@lists.debian.org` に投稿された内容と Debian Project News からです。

問題 15. Debian Project Leader からアシスタントに任命されたのは

- A Nobuhiro Iwamatsu
- B Hiroyuki Yamamoto
- C Luk Claes

問題 16. Debian Groupware Meeting が開催されたのはどこだったか

- A New York
- B Ogikubo, Suginami, Tokyo
- C LinuxHotel, Essen, Germany

問題 17. Debconf で 249 ユーロで配布される予定の携帯電話は？

- A Openmoko Neo Freerunner
- B iPhone
- C Android

問題 18. Debian Secretary Team にはっていないのは？

- A Kurt Roeckx
- B Neil McGovern
- C Kohei Maeda

問題 19. Andrew McMillan, Bill Allombert とともに Policy Team に追加されたのは

- A Colin Watson
- B Junichi Uekawa
- C Takuma Yamada

問題 20. DDTSS の URL は

- A `http://www.2ch.net/`
- B `https://launchpad.net/rosetta`
- C `http://ddtp.debian.net/ddtss/index.cgi/xx`

27 Debian Trivia Quiz 問題回答

上川 純一



- Debian Trivia Quiz の問題回答です。あなたは何問わかりましたか？
- | | |
|-------|-------|
| 1. C | 11. B |
| 2. B | 12. B |
| 3. A | 13. C |
| 4. B | 14. A |
| 5. C | 15. C |
| 6. B | 16. C |
| 7. A | 17. A |
| 8. C | 18. C |
| 9. B | 19. A |
| 10. A | 20. A |

『あんどきゅめんてっど でびあん』について

本書は、東京および関西周辺で毎月行なわれている『東京エリア Debian 勉強会』および『関西エリア Debian 勉強会』で使用された資料・小ネタ・必殺技などを一冊にまとめたものです。収録範囲は東京エリアは勉強会第 47 回から第 52 回、関西エリアは第 20 回から第 23 回まで。内容は無保証、つっこみなどがあれば勉強会にて。



あんどきゅめんてっど でびあん 2009 年夏号

2009 年 8 月 15 日 初版第 1 刷発行

東京エリア Debian 勉強会/関西エリア Debian 勉強会 (編集・印刷・発行)
