Grand Unified Debian

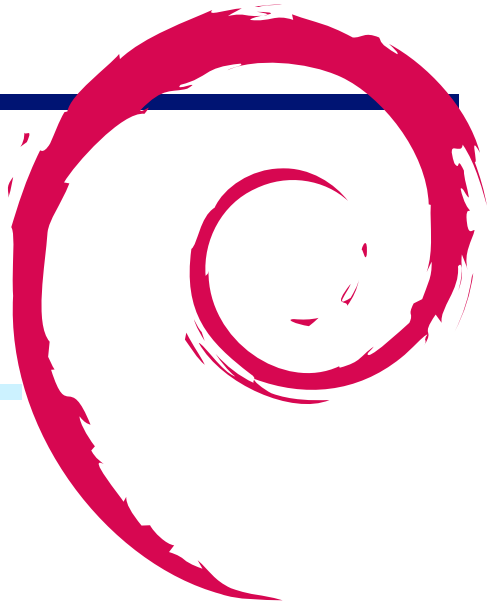
銀河系唯一のDebian専門誌

東京エリア/関西Ｄｅｂｉａｎ勉強会

あんどきゅめんてっど でびあん 2014 年夏号 2014 年 8 月 12 日 初版発行

デビアン勉強会

目次



1 Introduction

上川 純一, 山下 尊也

1.1 東京エリア Debian 勉強会

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 をがりがりと作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。

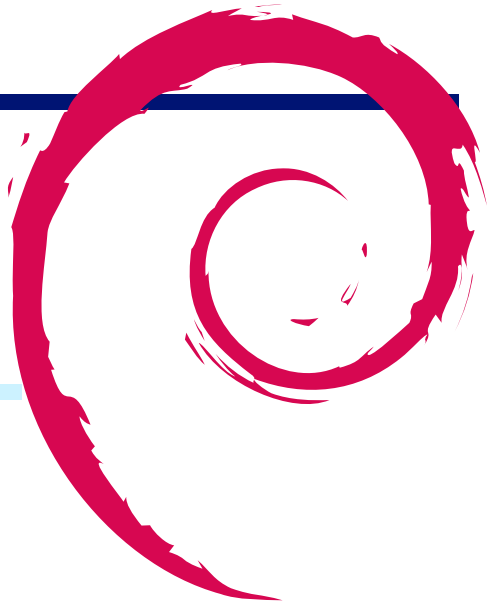
1.2 関西 Debian 勉強会

関西 Debian 勉強会は Debian GNU/Linux のさまざまなトピック (新しいパッケージ、Debian 特有の機能の仕組、Debian 界隈で起こった出来事、などなど）について話し合う会です。

目的として次の三つを考えています。

- メーリングリストや掲示板ではなく、直接顔を合わせる事での情報交換の促進
- 定期的に集まれる場所
- 資料の作成

それでは、楽しい一時をお楽しみ下さい。



2 Debian GNU/Hurd 2013

野島 貴英

2.1 はじめに

2013 年 5 月 21 日に、Debian GNU/Hurd 2013 がリリースされました [1]。今回は、この Debian GNU/Hurd 2013 を

- インストールしてみたり、
- 試したり
- 調べたり

した事を書いてみます。

2.2 インストールしてみる

Debian GNU/Hurd 2013 のインストール CD イメージは、Hurd ベースのネットワークインストール可能なイメージになっています。ここでは、実際に Linux の KVM を使い、Debian GNU/Hurd 2013 をインストールして動かしてみます。

なお、Debian GNU/Hurd 2013 は、i386 アーキテクチャが対象ですが、AMD64(64bit) 環境の KVM 上でそのまま何ら問題なく動作します。

Step 1. debian sid を用意し、過去の東京エリア Debian 勉強会の KDE 開発環境の資料 [1] を参考に、br0 デバイスをセットアップしておき、インターネットへアクセスできる環境を用意しておきます。

Step 2. Debian GNU/Hurd 2013 の NETINST CD イメージを入手します。

```
$ wget http://ftp.debian-ports.org/debian-cd/hurd-i386/current/debian-hurd-2013-i386-NETINST-1.iso
```

Step 3. KVM を使い、インストールします。コツとして、ディスク I/O は IDE、ネットワークデバイスは e1000 を利用するとよいです。

```
$ sudo aptitude install libvirt-bin virtinst
$ sudo qemu-img create -f raw /var/lib/libvirt/images/debian-hurd0 7G
$ sudo virt-install --connect=qemu:///system -n debian-hurd0 --ram 512 \
  --cdrom /home/yours/debian-hurd-2013-i386-NETINST-1.iso \
  --disk /var/lib/libvirt/images/debian-hurd0,bus=ide,size=7,format=raw,cache=writeback \
  --bridge=br0,model=e1000 --vnc --hvm --accelerate
```

また、インストーラで訊かれる質問は表 1 のようにしています。

なお、インストール中”Select and install software” メニューで、“Debian desktop environment” を指定していると、インストーラが途中で異常終了してしまいます。その時は諦めて、一旦”Debian desktop environment”

項番	項目名	値	備考
1	country	other → Asia → Japan	
2	Configure locales	United States en_US.UTF-8	
3	Configure the keyboard	おつかいのキーボードにて	106 キーは無い
4	NetworkConfigure Manually	192.168.0.2 等	お使いの環境にて
5	Partition disk	”Guided - use entire disk”	簡易的にこちらを選択。
6	mirror country	Japan 指定の、ftp.jp.debian.org を選択	

表 1　インストーラでの質問への回答例

をインストール候補から外してみて、Step 3. からやり直しとなります。[*1]

Step 4.　インストールが完了すると、勝手にブートして virt-viewer にて GNU/Hurd が立ち上がり、”login:” プロンプトが出ますので、ログインすると使えます。

2.3　使うにあたってちょっと知ると良いこと

基本的には UNIX 系システムの使い勝手です。Debian GNU/Linux 使える方なら、非常にとっつき易い感じです。もちろんですが、Debian なので、dpkg/apt はそのまま使えます。

ただ、hurd を使うにあたって、いくつか Linux とは違う点があるので、知ると便利な件をいくつか以下に記載します。

1. システム停止
   Linux システムですと、/sbin/shutdown　-h now とか、CTL+ALT+DEL のキーアクションとかが一般的かと思いますが、hurd の場合は sync;halt となります。shutdown を hurd で利用すると解るのですが、init と連携できなくて shutdown コマンドが途中で失敗します。
2. ネットワーク関係の設定
   Linux システムですと、ip コマンドとか、ifconfig コマンドとかありますが、hurd ですと settrans コマンド使って、/hurd/ディレクトリ以下のネットワーク用の translator というコマンド群と、特定のデバイスファイルを結びつけるという事で対応します。

   ```
   NIC 設定の例:
   # settrans -fgcap /servers/socket/2 /hurd/pfinet -i eth0 \
        -a 192.168.0.5 -m 255.255.255.0 -g 192.168.0.1
   ```

   また、hurd の場合、translator へ fsysopts コマンドで指示を出すことにより、translator が対応していれば設定変更もできます。

   ```
   NIC の設定がどうなっているか？
   # fsysopts /servers/socket/2
   /hurd/pfinet --interface=/dev/eth0 --address=10.3.0.1 --netmask=255.255.0.0 --gateway=10.3.0.128
   (ps -auxww | fgrep pfinet とかしても可)
   NIC 設定変更の例:
   # fsysopts /server/socket/2 -a 10.3.0.2 -m 255.255.0.0 -g 10.3.0.128
   ```

3. ファイルシステムのマウント
   Linux だと mount コマンドですが、hurd の場合ですと、

[*1] 自分がやった時は、xserver-xorg-video-all のパッケージ依存関係が満たせなかった様です。今は治っているかもしれません。

```
物理デバイスをマウント
# settrans /mnt /hurd/ext2fs /dev/hd0s5
CD イメージをマウント
# settrans /mount/point /hurd/iso9660fs CD_image.iso
NFS をマウント
# settrans -cgap /mount/point /hurd/nfs 192.168.1.1:/home
```

という形で、ファイルシステム用の translator を settrans コマンドでマウントポイントに結びつけてしまうという手を使います。

その他については、Debian GNU/Hurd Configuration(`http://www.debian.org/ports/hurd/hurd-install`)を見ると良いです。

2.4 X を動かしてみる

ログインしてそのまま CUI で利用という漢(おとこ)な使い方も確かにできますが、X ぐらいは動かしたい場合もあるわけです。試しに動かしてみます。

```
# aptitude install xserver-xorg-video-cirrus xinit fluxbox
# xinit
ここで、白いウィンドウが出るので、そのウインドウにカーソルを合わせ、
# fluxbox &
```

慣れてきたら、/root/.xinitrc とかにいろいろ書くと便利です。

なお、今回 X を root で動かしています。本来、一般ユーザで動かせると良いのですが、自分はまだしっかり調べきれていません。

2.5 GNU Hurd 図解

GNU Hurd を図解してみます。

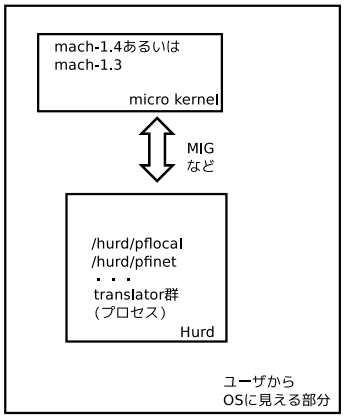


図 1　OS の図解

図のとおり、

- カーネル本体は mach-1.3/1.4
- ファイルシステム、ネットワークインターフェースなどは/hurd/以下にある translator と呼ばれる実行バイナリによるユーザプロセス
- カーネル本体と translator 群は主に MIG と呼ばれる RPC などで通信

という構造になっています。

2.6 終わりに

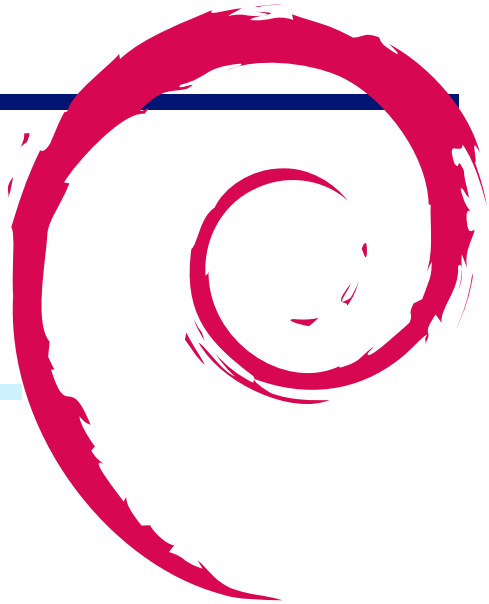
Debian GNU/Hurd は、まだいろいろと未開拓な部分も多いです。また、gnu hurd 本体もいろいろと他に機能が必要な状態です。

すでにいろいろと完成された Debian GNU/Linux も面白いですが、いろいろ未完成な Debian GNU/Hurd も Hack して遊ぶには面白いと思います。皆様もぜひ。

参考文献

[1] Debian.org,“Debian GNU/Hurd 2013 リリース!”, http://www.debian.org/ports/hurd/hurd-news

[2] 野島 貴英,「Debian 開発者の KDE 環境あれこれ」, 第 85 回東京エリア Debian 勉強会資料, http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume201202.pdf



3 Debian Pure Blend

野島 貴英

3.1 Debian Pure Blend

Debian に用意されている大量のパッケージをうまく使い、そのままの Debian のリポジトリを用いる事で、特定分野向けのシステムを容易にセットアップできるようにした Debian の仕組み（考え方）となります。

Debian パッケージに含まれる Recommends 情報が利用されて、指定された特定用途のパッケージが導入される仕組みとなっています。

3.2 用語

Debian Pure Blend を語る時に使われる用語を以下に載せておきます [1]。

項番	呼び名	概要	備考
1	Debian Pure Blend	そのままの Debian を用いて特定用途向けの Debian を実現	DebiChem, Debian Edu 等
2	Debian Blend	一部の Debian では未だ公式には採択されていないちょっとした変更を付け足し、残りはそのままの Debian を用いて特定用途向けの Debian を実現	
3	Debian Derivative	Debian 派生物と日本語では言われる。目的は様々であり、Debian を元にした新しいディストリビューションを作るという点では共通。Debian をベースにしたかもしれないが、現在では大量の変更/新規機能を加えて作られているディストリビューション。	ubuntu, SteamOS
4	web sentinel	`http://blends.debian.org`。PureBlend の種類と搭載されているパッケージ及び開発状況を載せているページ。	

表 2 用語

3.3 利用できる Pure Blend

現在 Debian unstable で見つかった Pure Blend 用のパッケージの一覧を載せます。

なお、経緯を追いかけきれていないのですが、”Existing Debian Pure Blends” に記載されている他の Blend のパッケージ [2] を自分は見つけることができませんでした。

項番	用途	PJ 名前	概要	メタパッケージ名
1	子供用	Debian Junior	子供向けの Debian を作る	junior-*
2	医療	Debian Med	医療関係者向けの Debian を作る	med-*
3	学校	Debian Edu	学校の情報教育向けの Debian を作る	education-*,debian-edu-*
4	科学技術	Debian Science	科学技術向けの Debian を作る	science-*
5	マルチメディア	Debian Multimedia	マルチメディア作成関係者向けの Debian を作る	multimedia-*
6	地理情報	DebianGIS	地理関係者向けの Debian を作る	gis-*
7	化学	Debichem	化学関係者向けの Debian を作る	debichem-*
8	中国語対応の一例	Debian EzGo	中国語に対応した Debian の 1 つを作る	ezgo-*

表 3　Debian unstable で見つかる Pure Blend

3.4　使ってみる

早速使ってみます。ここでは、Debian Junior を選んでみます。

Step 1.　debian を用意し gnome desktop を導入しておきます。

Step 2.　Debian Junior のメタパッケージの一つである、junior-gnome を入れてみます。

```
$ sudo aptitude install junior-gnome
... junior-gnome/compris/gworldclock/mathwar が導入される...
```

gnome のメニューを見ると、gcompris/gworldclock/mathwar が新規に増えている事が分かります。

3.5　仕組み

Pure Blend のパッケージは、Recommends としてに導入したい特定用途向けのアプリケーションが指定されています。そのため、Pure Blend のパッケージを導入すると、Recommends に指定されたアプリケーションがまとめて導入されます。

試しに、先の例の junior-gnome の依存関係を調べてみます。

```
$ apt-cache depends junior-gnome
  Depends: junior-tasks
  Depends: junior-config
  Recommends: gcompris
  Recommends: gworldclock
  Recommends: mathwar
```

Recommends として、gcompris/gworldclock/mathwar が指定されている事がわかります。

3.6　Pure Blend 用のパッケージを作ってみる

Pure Blend 用のパッケージを試しに作ってみます。blends-dev パッケージを導入することで、task ファイルから Pure Blend 用のパッケージを簡単に作る事ができます。

ここでは、例として、ゲームのジャンルで visual novel を選び、これらのパッケージを導入するようなパッケージを作成してみます。

Step 1.　blends-dev パッケージを導入します。

項目	内容	説明
PJ 名	Debian-visualnovel	visualnovel 用途向け
パッケージ名	visualnovel-*	visualnovel メタパッケージ
tasksel 用パッケージ	visualnovel-task	tasksel 用定義ファイル

表 4　今回の Pure Blend の定義

```
$ sudo aptitude install blend-dev
```

Step 2.　作業ディレクトリを作り、必要なファイルを blends-dev パッケージに梱包されているテンプレートをコピーして用意します。

```
$ mkdir debian-visualnovel-1.0
$ cd debian-visualnovel-1.0
$ cp -a /usr/share/doc/blends-dev/examples/config .
$ cp -a /usr/share/doc/blends-dev/examples/debian .
$ cp -a /usr/share/doc/blends-dev/examples/tasks .
$ cp /usr/share/doc/blends-dev/examples/Makefile .
$ cp /usr/share/doc/blends-dev/examples/README .
```

Step 3.　コピーした各ファイルに含まれる”_BLEND_” の文字列を”visualnovel” に全部置き換えます。

```
$ find . -type f | xargs -n 1 sed -i 's/_BLEND_/visualnovel/g'
```

Step 4.　debian/control.stub を編集します。ここで、Package の定義として、visualnovel-tasks を必ず追加指定しておきます。また、debian/changelog も書いておきます。

```
$ vi debian/control.stub
----debian/control.stub ここから-------
Source: debian-visualnovel
Section: misc
Priority: extra
Maintainer: Your Name <Your mail>
Build-Depends-Indep: debhelper (>= 9),
                     blends-dev (>= 0.6.16.4)
Standards-Version: 3.9.4

Package: visualnovel-tasks
Architecture: all
Depends: tasksel
Description: Debian visualnovel for tasksel
 This package provides Debian visualnovel tasks in tasksel.

----debian/control.stub ここまで-------
$ vi debian/changelog
----debian/changelog ここから------
debian-visualnovel (1.0) unstable; urgency=low

  * initial release

 -- Your Name <your@e-mail>  Mon, 13 Jan 2014 23:56:00 +0900

----debian/changelog ここまで------
```

Step 5.　tasks/task1 を tasks/game に変更し、どのパッケージを導入するか等を Depends:行に指定します。visual novel ですので、onscripter と、renpy-thequestion を入れてみます。

```
$ mv task/task1 task/game
$ vi task/game
--------task/game ここから----------
Task: game
Description: Debian-visual novel games_
 This metapackage will install Debian packages for use in
 game of Debian-visualnovel

Depends: onscripter, renpy-thequestion

--------task/game ここまで----------
```

Step 6.　パッケージを作るために必要なファイルを自動生成します。

```
$ make -f debian/rules gen-orig-source
```

Step 7. パッケージをビルドします。

```
$ debuild
...visualnovel-game/visualnovel-task パッケージが出来上がる...
```

無事 visualnovel-*パッケージができました。

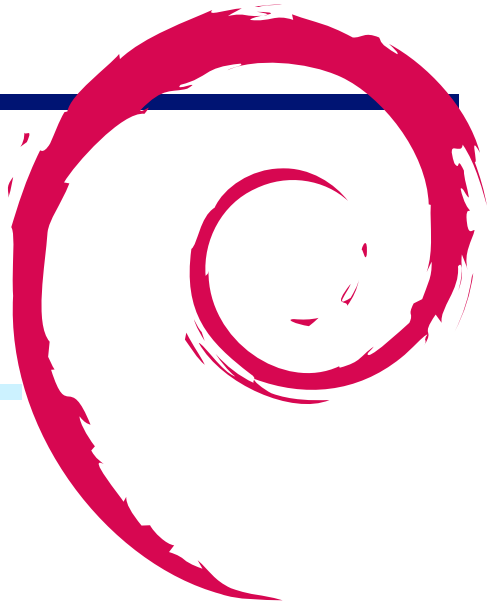
3.7 おわりに

今回は、Debian Pure Blend の導入から作成まで説明してみました。豊富なパッケージを持つ Debian ならではの考え方と思います。これを機に、皆さんも何か Debian Pure Blend を作ってみませんか?

参考文献

[1] Debian wiki,“DebianPureBlends”, https://wiki.debian.org/DebianPureBlends

[2] blends.debian.org,”Existing Debian Pure Blends”, http://blends.debian.org/blends/ch04.html



4 debian で docker.io

野島 貴英

4.1 はじめに

去年あたりから、linux のコンテナ環境である docker[1] が注目されているようです。

docker は使うとわかるのですが、単に linux 上にコンテナ環境を作成するという機能の他に、aufs を利用してベースの OS のシステムに迅速に変更差分を適用するという動作により、非常に素早くカスタム化されたコンテナ環境の作成・変更・管理が出来ます。

また、変更した内容を docker リポジトリ (`https://index.docker.io`) に登録することにより、docker が動作する環境さえあれば、こちらのリポジトリから全く同じコンテナ環境を作成・動作させることができます。

今回は debian を docker ホストにして docker を使ってみた事について発表します。

4.2 今回利用の debian

今回発表で評価した debian は unstable(jessie/sid) となります。

なお、残念ながら安定版の debian wheezy では未だ docker はパッケージ化されていない状況です。本誌を読まれている debian 使いの方は、ぜひこの機会に debian unstable(jessie/sid) にアップグレードしてパッケージから docker をお試しください。

4.3 docker 仕組みの簡単なおさらい

docker を図示すると図のようになります。

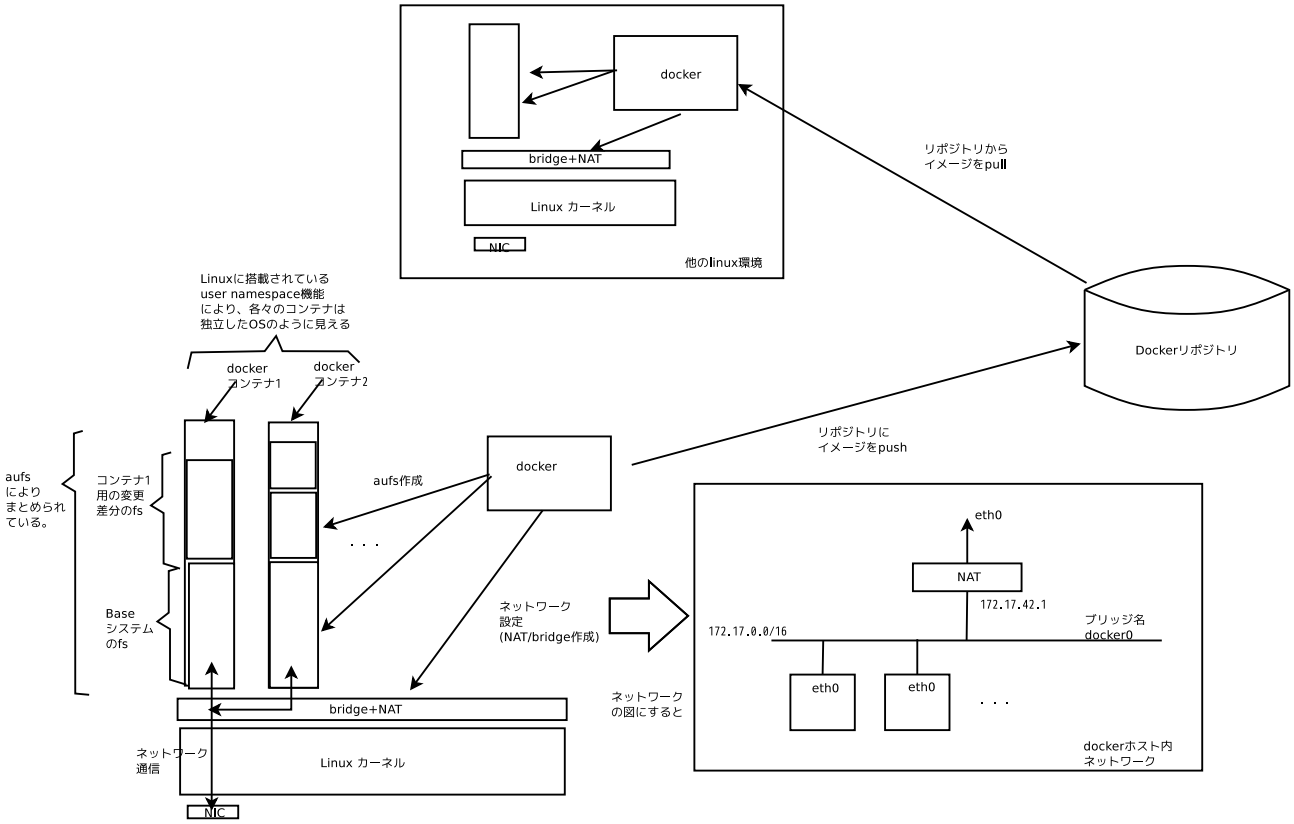


図 2　docker の構造

docker によるコンテナ環境の特徴としては、

- 非常に素早い起動、停止が可能です。
- base のファイルシステムに、aufs による差分ベースのファイルシステム内容の適用を行うため、非常に簡単にコンテナの変更・破棄が可能です。
- 構成管理を Docker リポジトリで行える。（注：一見 git のような使い方に見えますが、git を使って作られているわけではありませんので、git ほどの高機能で柔軟な変更管理はできません）
- Docker リポジトリが参照でき、docker が動く環境であれば、docker ホストの linux ディストリビューションが異なる環境でも同じ構成内容を持つ docker コンテナを動作できます。

となります。

docker ホストの内部のネットワークは、docker により、ブリッジ docker0 が作成され、docker ホストの eth0 へ NAT されて接続されます。そのため、docker ホスト外からのコンテナ側のサービスへのアクセスは、DNAT して docker ホスト側のポートへ引き出すことにより行われます。

4.4　手元の debian 機材で試す

以下の手順で簡単に試すことが出来ます。

Step 1.　インターネットに接続できている debian unstable 環境を用意します。

Step 2.　ip forwarding ができるようにしておきます。

```
$ sudo vi /etc/sysctl.d/ip-forward.conf
----ここから-----
net.ipv4.ip_forward=1
----ここまでを記載-----
$ sudo sysctl -p /etc/sysctl.d/ip-forward.conf
```

Step 3. docker を導入します。なお、debian パッケージの docker は、docker.io というパッケージ名であり、コマンドも docker ではなく、**docker.io** という名前になります。（以降本コマンドを docker.io コマンドと呼びます）

```
$ sudo aptitude install docker.io
$ docker.io
Usage: docker [OPTIONS] COMMAND [arg...]
 -H=[unix:///var/run/docker.sock]: tcp://host:port to bind/connect to or unix://path/to/socket to use

A self-sufficient runtime for linux containers.

Commands:
    attach    Attach to a running container
... 中略 (docker.io コマンドの help が出る)...
```

Step 3. グループ docker に操作者のログイン ID を追加し、ログインしなおします。こうすることで docker.io コマンドによる操作を一般ユーザ権限で操作できるようになります。

```
$ sudo useradd YOUR-ID docker
$ exit
login: YOUR-ID
Password: xxxxx
$
```

Step 4. 試しにコンテナとして debian-sid(jessie-sid) を docker で動かしてみます。

```
$ docker.io run -t -i -h debian-sid1 debian:sid
Unable to find image 'debian:sid' locally
Pulling repository debian
1cda8535c670: Download complete
511136ea3c5a: Download complete
0ed2a4d77969: Download complete
root@debian-sid:/#  <-- 起動した docker コンテナの debian sid。
```

Step 5. 作った docker コンテナの中で ps を導入し、process を見てみます。

```
root@debian-sid:/# apt-get install procps
root@debian-sid2:/# hash
hits        command
   2        /sbin/ip
   1        /usr/bin/apt-get
root@debian-sid2:/# ps -auxww
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  18016  1932 ?        Ss   22:17   0:00 /bin/bash
root       118  0.0  0.0  17488  1136 ?        R+   22:26   0:00 ps -auxww
root@debian-sid2:/#
```

見るとわかるとおり、init の代わりに/bin/bash が PID=1 で動作している状態です。

Step 6. Ctrl+p Ctrl+q を連続で打ち込むと、docker コンテナの shell から抜けます。なお、exit を実行すると、PID=1 の/bin/bash が終了するため、shutdown を実行したことと等価となり、コンテナが終了します。

```
root@debian-sid2:/# ... ここで Ctrl+p Ctrl+q する...
$ <- docker ホストのプロンプトが帰ってくる
$ docker.io ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
20fa6020f73b        debian:sid          /bin/bash           12 minutes ago      Up 12 minutes                           evil_euclid
(コンテナ ID: 20fa6020f73b が動作中であることを示す)
$ docker.io attach 20fa6020f73b (<--再び debian-sid に接続)
<リターンキー押す>
root@debian-sid2:/# <-再びコンテナの shell プロンプト。
root@debian-sid2:/# exit
$ docker.io ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
$
(docker.io ps をとって稼働中のコンテナを確認したが、コンテナが終了してしまっているため、動作中のコンテナ ID が表示されない↑)
```

4.5 docker のネットワークの癖について

docker は、docker ホストの起動時に docker がデーモンモードで動作しており、docker コマンドで指令を送ってコンテナの管理をします。ここで、docker ホストのネットワークを、デーモンモードの docker が起動時にセットアップしています。

ここで、例えば docker ホストがモバイル PC であった場合、ppp とかを後から起動するなどして、ネットワークの設定が docker デーモンがセットアップした状態と異なってしまうことがあります（特に NAT 周り。）

この場合、docker.io でコンテナを作成しようとしても、作成途中のコンテナ側からネットワークが外部へのネットワーク通信が出来ず、コンテナ作成が途中で停止する現象が起きることがあります。

この場合は、ppp などの通信をつないだ状態で、再度、docker デーモンを再起動すると、再度ネットワークがセットアップされ、問題が解決します。

```
$ pon xxxx <-- ppp を起動などして NAT が ppp の定義で書き換えられてしまう。
$ docker.io run -t -i -h debian-sid1 debian:sid
Unable to find image 'debian:sid' locally
Pulling repository debian
1cda8535c670: Download complete
... ここでハングアップしてしまう...
(Ctrl+C で停止させる)
$ sudo systemctl restart docker.io.service
(docker デーモンの再起動が行われる)
$ docker.io run -t -i -h debian-sid1 debian:sid
Unable to find image 'debian:sid' locally
Pulling repository debian
1cda8535c670: Download complete
511136ea3c5a: Download complete
0ed2a4d77969: Download complete
root@debian-sid:/#  <-今度はコンテナが無事起動する。
```

4.6 GCE で docker

手元の debian 機で docker を動かすだけでは物足りないかと思います。今年は immutable infrastructure[2] の年ですので、早速パブリッククラウド環境でも動かしてみることにします。

Google Compute Engine(GCE) でも docker は動くとのことですので、試してみます。

Step 1. お手元の debian sid で chromium などのブラウザを使い、google アカウントで google にログインしておきます。

Step 2. `https://developers.google.com/` の google デベロッパサイトから、Google Cloud にサインアップします。

注意：巷の blog などでは、`https://developers.google.com/compute/docs/signup` が案内されていますが、評価期間は終わっているため、こちらからサインアップする必要はありません。長い英語のアンケートに英語で答えさせられるなどの苦行が待っているため、こちらはおすすめしません。

Step 3. プロジェクトを作成するメニューが最初に現れますので、適当にプロジェクトを作成します。

Project Name: docker evalation

Project ID: docker-evaluation-test-001

Step 4. billing(支払い) メニューになるので、支払いの情報を記載します。

Step 5. 特に折り返しの電話などなく、GCE のメニューになります。

Step 6. お手元の debian unstable 機材に、google-cloud-sdk をダウンロードします。

```
$ mkdir google-sdk;cd google-sdk
$ wget https://dl.google.com/dl/cloudsdk/release/google-cloud-sdk.tar.gz
$ tar xzf google-cloud-sdk.tar.gz
$ cd google-cloud-sdk
$ ./install.sh
 ... カレントディレクトリにインストールが開始...
$ cd ..
$ zsh の場合：source google-cloud-sdk/path.zsh.inc;rehash
  bash の場合:source google-cloud-sdk/path.bash.inc;hash
```

Step 7. sdk から認証を行います。

```
$ gcloud auth login
... ここで、chromium などが開き、sdk が google のアカウントにアクセスしてよいかの
    許可を求められるので、「承諾」を押下...
$
```

Step 7. GCE の instance を作ります。ここでは、料金の最も安い f1-micro で、ネットワーク的に近いアジア地域に、debian wheezy のイメージで作ります。3 秒ぐらいで完了します。

```
$ gcutil addinstance docker-test001 --project=docker-evaluation-test-001 --image=debian-7 --machine_type=f1-micro \
--zone=asia-east1-a --wait_until_running --auto_delete_boot_disk
INFO: Resolved debian-7 to projects/debian-cloud/global/images/debian-7-wheezy-v20140415
INFO: Waiting for insert of instance docker-test001. Sleeping for 3s.
INFO: Waiting for insert of instance docker-test001. Sleeping for 3s.
INFO: Waiting for insert of instance docker-test001. Sleeping for 3s.
INFO: Waiting for insert of instance docker-test001. Sleeping for 3s.
INFO: Ensuring docker-test001 is running.  Will wait to start for: 240 seconds.
  ... 中略...
 $
```

Step 8. 作ったインスタンスにログインして、早速 debian unstable にアップグレードします。

```
$ gcutil ssh --project=docker-evaluation-test-001 docker-test001
yours@docker-test001:~$ sudo vi /etc/apt/source.list
--------------全部消して以下に置き換え--------------------
deb     http://gce_debian_mirror.storage.googleapis.com/ sid         main contrib non-free
deb-src http://gce_debian_mirror.storage.googleapis.com/ sid         main contrib non-free
deb     http://http.debian.net/debian sid         main contrib non-free
deb-src http://http.debian.net/debian sid         main contrib non-free
--------------全部消して以下に置き換えここまで--------------------
yours@docker-test001:~$ sudo apt-get update
yours@docker-test001:~$ sudo apt-get dist-upgrade
... ここで、grub はインストールせずに進めるを選択...
yours@docker-test001:~$ dpkg-reconfigure locales
...en_US.utf-8 と、ja_JP.utf-8 をメニューで選択して有効にする。
yours@docker-test001:~$ exit
$ gcutil restart --project=docker-evaluation-test-001 docker-test001
... リスタートしてしばらく待つ...
$ gcutil ssh --project=docker-evaluation-test-001 docker-test001
yours@docker-test001:~$ uname -a
Linux docker-test001 3.14-1-amd64 #1 SMP Debian 3.14.4-1 (2014-05-13) x86_64 GNU/Linux
yours@docker-test001:~$ cat /etc/debian_version
jessie/sid
```

Step 9. 早速 GCE インスタンスに docker.io パッケージを導入し、下準備。

```
yours@docker-test001:~$ sudo vi /etc/sysctl.d/ip4forward.conf
-----ここから--------
net.ipv4.ip_forward = 1
-----ここまで--------
yours@docker-test001:~$ sudo sysctl -p /etc/sysctl.d/ip4forward.conf
net.ipv4.ip_forward = 1
yours@docker-test001:~$ aptitude install docker.io
yours@docker-test001:~$ sudo useradd yours docker
yours@docker-test001:~$ exit
... 再度ログインしなおし...
$ gcutil ssh --project=docker-evaluation-test-001 docker-test001
yours@docker-test001:~$
```

Step 10. docker を動かしてみる。

```
yours@docker-test001:~$   docker.io run -t -i -h debian-sid debian:sid
Unable to find image 'debian:sid' locally
Pulling repository debian
1cda8535c670: Pulling image (sid) from debian, endpoint: https://cdn-registry-1.docker.io/v1cda8535c670: Download complete
511136ea3c5a: Download complete
0ed2a4d77969: Download complete
root@debian-sid:#   <---無事動作
```

GCE でも非常に簡単に動作しました。

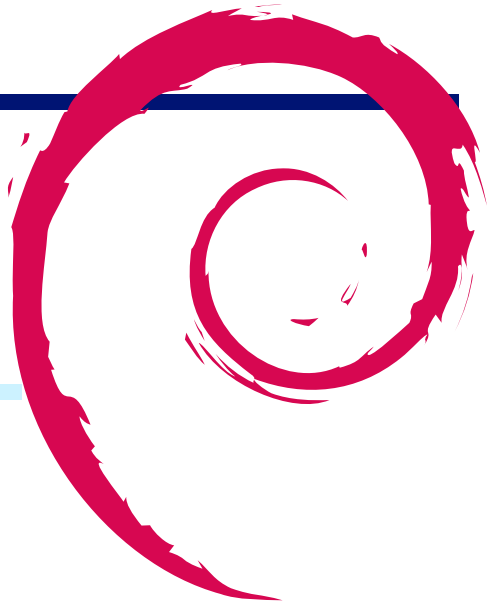
4.7 終わりに

docker を debian で動かす事について述べました。見ての通り非常に簡単に動かすことができます。また、docker は docker コンテナの起動・停止・再作成が非常に早く、軽快に使うことができます。今は docker は勢いがあるので、皆さんもぜひ評価されてはいかがでしょうか？

参考文献

[1] “Docker: the Linux container engine”, https://www.docker.io/

[2] publickey,「Immutable Infrastructure はアプリケーションのアーキテクチャを変えていく」, http://www.publickey1.jp/blog/14/immutable_infrastructure_1.html



5 Debian で dnsmasq を使う

野島 貴英

5.1 はじめに

dnsmasq とは、pxe/bootp/dhcp/tftp/dns フォワーダ/dns サーバーを一手に引き受ける事ができる便利かつ小さなソフトウェアです。こちらがあれば、仮想環境を使った debian システムのデバッグ、複数の debian 環境が必要となるような開発環境構築などにとても便利です。

ここでは、debian にて、dnsmasq を導入し、いろいろな使い方をしてみます。

5.2 前提知識

過去の東京エリア Debian 勉強会の KDE 開発環境の資料 [1] を見ておくとスムーズです。こちらに、本発表にて多数出てくる、br0 デバイスをセットアップについて記載しています。

5.3 DNS フォワーダとして使う

ノート PC に入れた Debian 上にて、KVM/bridge を使い、複数ホストで構成される LAN 環境を作って何か開発をすることを想定します。当然ノート PC はモバイル利用が主になりますので、モバイル用の通信アダプタを繋いだり、free wifi スポットのある喫茶店へ行ったり、東京エリア Debian 勉強会でハックしたり、有線繋いだりと、いろいろなグローバル回線のある環境へ接続できる必要があります。

ここで問題になるのは環境によって様々に変わる DNS リゾルバの扱いとなります。仮想環境上の Debian の DNS リゾルバのアドレスは、ノート PC をどこに繋ぐかによらず、一定のアドレスであると、とても便利です。こういった環境を作るのに dnsmasq は便利です（図 3、図 4 参照。）

1. DNSフォワーダが無い場合

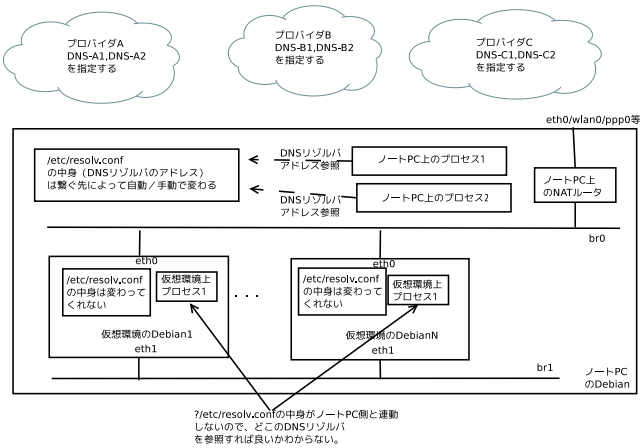


図 3　dns フォワーダが無い場合

2. DNSフォワーダが有る場合

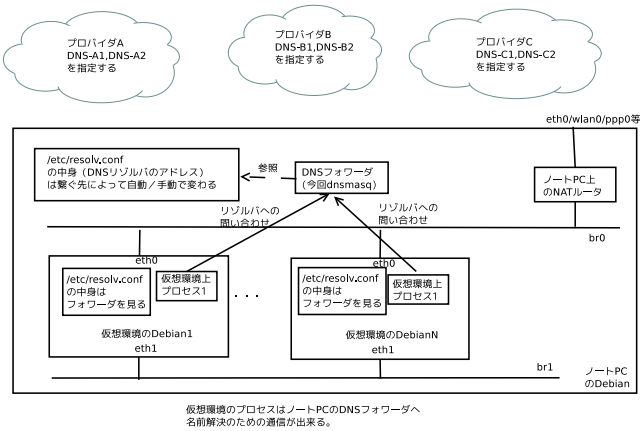


図 4　dns フォワーダがある場合

では早速使ってみます。

Step 1.　dnsmasq を導入します。

```
note-pc$ sudo aptitude install dnsmasq
```

Step 2.　br0 のみ listen するようにし、dns フォワーダとしてのみ動作するように設定します。

```
note-pc$ sudo vi /etc/dnsmasq.d/forwarder.conf
interface=br0
no-dhcp-interface=br0
bind-interfaces
```

Step 3.　dnsmasq をリスタートします。

```
note-pc$ sudo service dnsmasq restart
```

Step 4.　動作を確かめます

```
特定のポートだけで Listen していることを確かめる
note-pc$ sudo netstat -nlp | fgrep dnsmasq
tcp  0 0 127.0.0.1:53  0.0.0.0:*   LISTEN      16955/dnsmasq
tcp  0 0 192.168.0.1:53  0.0.0.0:* LISTEN      16955/dnsmasq
udp  0 0 127.0.0.1:53    0.0.0.0:*             16955/dnsmasq
udp  0 0 192.168.0.1:53  0.0.0.0:*             16955/dnsmasq
実際に dnsmasq を指定して名前を引いてみる
note-pc$ sudo aptitude install dnsutils
note-pc$ dig @192.168.0.1 www.debian.org a
; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Debian-1:9.9.3.dfsg.P2-4 <<>> @192.168.0.1 www.debian.org a
; (1 server found)
;; global options: +cmd
... 中略...
;; ANSWER SECTION:
www.debian.org.                300        IN        A        5.153.231.4
www.debian.org.                300        IN        A        128.31.0.51
www.debian.org.                300        IN        A        130.89.148.14
... 中略...
```

非常に簡単に DNS フォワーダとしてセットアップ出来ます。

また、グローバル回線を変更した場合（例：拠点 A の wifi スポットから、拠点 B の wifi スポットへ移動した等）は、

```
... グローバル回線を有効にして...
note-pc$ sudo service dnsmasq restart
```

するだけで、新しいリゾルバ先を dnsmasq は取り込み、名前解決が出来るようになります。

5.4 簡易 DNS サーバーとして使う

今度は単一のドメインのホストレコードを返す簡単な DNS サーバーがちょっと欲しくなる時があります。このような場合、dnsmasq は簡易 DNS サーバーとしてもあっさり動作させることができます。

実は dnsmasq はデフォルトで/etc/hosts を読み込み、DNS サーバーとしても動作しています。そのため、簡易 DNS サーバーとして利用する場合、/etc/hosts をそのまま利用するのが一番簡単です。

Step 1. /etc/hosts に名前解決したいホストを書きならべる。

```
#例となります。
note-pc$ sudo vi /etc/hosts
-----------/etc/hosts ここから---------------
192.168.0.3 debian0
192.168.0.4 debian1.my-domain debian1
192.168.0.5 debian2.my2-domain debian2
-----------/etc/hosts ここまで---------------
```

Step 2. dnsmasq を restart します。

```
note-pc$ sudo service dnsmasq restart
```

Step 3. 実際に DNS を引いて確かめてみます。

```
note-pc$ dig @192.168.0.1 debian0 +short
192.168.0.3 (←正しく返却されている)
note-pc$ dig @192.168.0.1 debian1.my-domain +short
192.168.0.4 (←正しく返却されている)
note-pc$ dig @192.168.0.1 debian2.my2-domain +short
192.168.0.5 (←上とは異なるドメイン所属のレコードも正しく返却されている)
note-pc$ dig @192.168.0.1 debian2 +short
192.168.0.5 (←ホスト名だけでも正しく返却されている)
```

5.5 簡易 PXE ブートをさせてみる

dnsmasq を使って pxe ブートを仮想化環境である KVM から行ってみます。ここでは、文献 [2] の方法を使い、実際に wheezy の netinst 用インストーラが立ち上がるまで確かめてみます。

Step 1. /etc/dnsmasq.d/にて、今まで書いたファイルを一旦消去 (適宜)

```
note-pc$ sudo rm -f *
```

Step 2. pxeboot 用の定義を記載

```
note-pc$ sudo vi /etc/dnsmasq.d/pxeboot.conf
----- pxeboot.conf の中身-----
interface=br0
bind-interfaces
dhcp-range=192.168.0.129,192.168.0.254,255.255.255.0,1h
dhcp-boot=pxelinux.0,pxeserver
pxe-service=x86PC, "Install Linux", pxelinux
enable-tftp
tftp-root=/home/your/srv/tftp
----- pxeboot.conf の中身-----
```

Step 3. pxe ブートさせるイメージを展開しておく。

```
note-pc$ cd /home/your/
note-pc$ mkdir srv;mkdir tftp
note-pc$ cd srv/tftp
note-pc$ wget http://ftp.debian.or.jp/debian/dists/stable/main/installer-amd64/current/images/netboot/netboot.tar.gz
note-pc$ tar xzf netboot.tar.gz
```

Step 4. dnsmasq を再起動

```
note-pc$ sudo service dnsmasq restart
```

Step 5. virt-install コマンド経由で、pxe ブートしてみる。

```
note-pc$ sudo qemu-img create -f raw /var/lib/libvirt/images/debian-pxe 5G
note-pc$ sudo virt-install --connect=qemu:///system -n debian-pxe --ram 512 \
           --pxe --disk /var/lib/libvirt/images/debian-pxe,bus=virtio,size=5,format=raw,cache=writeback \
           --bridge=br0,model=virtio --vnc --hvm --accelerate
```

virt-viewer がすぐに開き、PXE ブートして、Debian wheezy のインストール画面が出てきます。

5.6 終わりに

今回ここでは、簡易 dns フォワーダ、簡易 dns サーバー、簡易 pxe サーバーを dnsmasq を使って組み立てました。

しかし、man dnsmasq を読むと判る通り、もっと複雑な事も簡単に出来ます。また、自分はまったく未評価ですが、最近では lua 言語と合わせて使う事ができるようです。

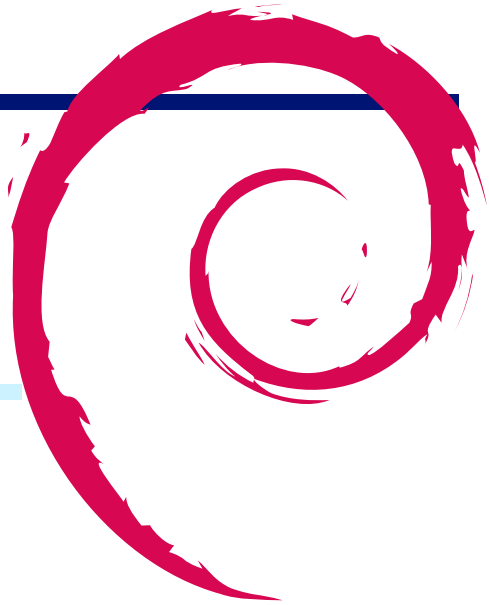
複数の仮想環境を使ってノート PC 上に複数の Debian の開発環境を立ち上げる場合に、非常に便利です。dnsmasq をぜひ一度お試しあれ。

参考文献

[1] 野島 貴英,「Debian 開発者の KDE 環境あれこれ」, 第 85 回東京エリア Debian 勉強会資料, http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume201202.pdf

[2] Debian.org,“PXEBootInstall”, https://wiki.debian.org/PXEBootInstall

201403 tokyo



6 Debian で iphone5 を繋ぐ

野島 貴英

6.1 はじめに

大変不自由なスマートフォンなのに日本で驚異的に売れまくっているという、目を被いたくなるような現実を作り出しているスマートフォンの 1 つとして、iphone5 があります [1]。このスマートフォン、PC に繋ぐには、windows/mac で iTune などの専用プロプリエタリなソフトウェアを使ってデータ同期をしなければならないという、これでもかというぐらいの不自由仕様となっています。

ここでは、少しでも iphone5 の不自由さを回避するため、

- 自由な Debian マシンに、なんとかして不自由な iphone5 を繋ぐ事について、
- iphone5 がどのような仕組みでつながるのか（Debian 開発者向け）

について述べます。

6.2 本発表内容についての情報ソース

本発表内容の技術の情報ソースは、すべて

- Apple 社のディベロッパーサイトで公開情報となっているもの (文献 [2])
- Debian パッケージに含まれるプログラムを解析した範囲
- その他 Web にて公開されている情報 (文献 [3],[4], [5])

のみに基づきます。

6.3 Debian に繋ぐ

東京エリア Debian 勉強会にいらっしゃるような方々は、普段から Debian sid をお使いかと思いますので、ここでは、Debian sid を用い、さらにパッケージのバージョンが低くて問題のある部分だけちょっと自作して[*2]繋ぐ方法を取ってみます。

まずは、用意するものを表 5 に記載します。

Step 1. iphone5 に Document 2 Free[*3]をインストールしておきます。

Step 2. debian sid で導入される libmobiledevice4 の upstream バージョンが古く、iOS7.1 に対応出来ないので、upstream 最新版からパッケージを自作して最新のものにします。

[*2] すみません、bug report あげときます...

[*3] https://itunes.apple.com/us/app/documents-2-free-file-manager/id314894105

項番	品目	備考
1	Debian sid の入っているマシン	
2	iphone5	iOS7.1（3/14 現在最新）にバージョンアップ済み
3	Lightning-USB ケーブル	
4	Debian と繋ぐ先の iphone アプリ。なんでも良いかと思いますが、ここではファイルマネージャアプリである Document 2 Free を例にあげます。	

表 5　用意するもの

```
$ sudo aptitude install git
$ git clone https://github.com/libimobiledevice/libimobiledevice.git libimobiledevice-1.1.6
$ mkdir libimobiledevice4
$ cd libimobiledevice4
# Debian で用意されている libimobiledevice に梱包されている debian ディレクトリを利用
$ apt-get source libimobiledevice4/sid
$ cd libimobiledevice-1.1.5
$ cp -a debian ../../libimobiledevice-1.1.6
$ cd ../../libimobiledevice-1.1.6
# いろいろパッチを当てる
# 主に、libimobiledevice-1.1.6 だと 1.1.5 用のパッチは upstream で適用済みなのではずす、
# doc パッケージの中身は作り方が違う、一部シンボルテーブルも違うなどで
# チェックを除外するのが目的。
$ rm -f debian/libimobiledevice-doc.doc-base
$ rm -f debian/libimobiledevice-doc.install
$ rm -f debian/libimobiledevice-doc.links
$ rm -f debian/libimobiledevice4.symbols
$ rm -rf debian/patches
```

```
$ patch -p1 <<__HERE
diff -ur a/debian/changelog b/debian/changelog
--- a/debian/changelog  2013-10-30 02:42:24.000000000 +0900
+++ b/debian/changelog  2014-03-13 21:50:16.000000000 +0900
@@ -1,3 +1,9 @@
+libimobiledevice (1.1.6-1~a1) unstable; urgency=low
+
+  * update latest upstream
+
+ -- Your Name <your@mail.addr>  Fri, 28 Feb 2014 01:42:21 +0900
+
 libimobiledevice (1.1.5-2) unstable; urgency=low

   * [0052e46] Drop hal fdi file.
diff -ur a/debian/control b/debian/control
--- a/debian/control    2013-10-30 02:42:24.000000000 +0900
+++ b/debian/control    2014-02-28 01:42:09.000000000 +0900
@@ -102,13 +102,3 @@
  .
  This package contains utilities and examples which use libimobiledevice.

-Package: libimobiledevice-doc
-Architecture: all
-Section: doc
-Depends: libjs-jquery, ${misc:Depends}
-Description: Library for communicating with iPhone and iPod Touch devices
- libimobiledevice is a library that talks the native Apple USB protocols that
- the iPhone and iPod Touch use. Unlike other projects, libimobiledevice does
- not depend on using any existing libraries from Apple.
- .
- This package contains the documentation for the library.
diff -ur a/debian/rules b/debian/rules
--- a/debian/rules      2013-10-30 02:42:24.000000000 +0900
+++ b/debian/rules      2014-02-28 01:49:58.000000000 +0900
@@ -25,7 +25,7 @@
        rm -rf $(CURDIR)/debian/tmp//usr/lib/python*/dist-packages/*.a
        #Remove installed man pages, installed by *.manpages
        rm -f $(CURDIR)/debian/tmp/usr/share/man/man1/*.1
-       dh_install --fail-missing
+       dh_install

 override_dh_strip:
        dh_strip --dbg-package=libimobiledevice4-dbg
@@ -34,5 +34,3 @@
        # Only build for the current version of python, not all supported.
        dh_python2 --no-guessing-versions

-override_dh_makeshlibs:
-       dh_makeshlibs -- -c4
__HERE
$ sudo aptitude build-dep libimobile4/sid
$ debuild -uc -us
$ cd ..
$ sudo dpkg -i ./libimobiledevice4_1.1.6-1~a1_amd64.deb ./libimobiledevice-utils_1.1.6-1~a1_amd64.deb
```

Step 3. 他に iphone5 に繋ぐための必要なパッケージを導入しておきます。

```
$ sudo aptitude install ideviceinstaller ifuse
... いろいろ依存関係に引きずられて入る...
```

Step 4. Debian マシンにて、fuse グループにユーザを追加し、一旦ログアウト＆ログイン操作を行います。（ログアウト＆ログイン操作を行う事が重要です）

```
$ sudo usermod -a -G fuse <your-login-id>
$ exit（あるいは、デスクトップ環境のログアウト操作）
your machine login: <your-login-id>
Password: ...your pass...
（あるいは、デスクトップ環境のログイン操作）
$
```

Step 5. iphone5 のロック画面を解除し、Lightning-USB ケーブルで Debian マシンと繋ぎます。

Step 6. Debian 側、iphone5 側両方に「コンピュータ/デバイスを信頼するか？」という意味のポップアップが表示されるかもしれませんが、「信頼する」を押して一旦抜けます。

Step 7. iphone5 とペアリングを行います。

```
$ sudo idevicepair pair
SUCCESS: Paired with device ...40 桁の uuid...
```

なお、この時、iphone5 に「コンピュータを信頼しますか？」というポップアップが出るので、「信頼する」を選択します。すると、Debian 機材の/var/lib/lockdown/以下に必要ファイルができる。以降は idevicepair 操作を行わなくても iphone5 との通信が出来るようになります。

Step 8. iphone5 にすでにインストールされているアプリケーションの appid を Debian から探します。こちらの結果から、Document 2 Free の appid は com.savysoda.documents2Free である事が分かります。

```
$ ideviceinstaller -l
Total: 16 apps
com.savysoda.documents2Free - Documents 2 7.3
com.square-enix.gunsandsouls - GUNS 1.0.1
com.google.b612 - Google Earth 7.1.1
... 中略...
```

Step 9. Document 2 Free のストレージをマウントし、本勉強会資料を投げ込んでみます。(もちろん、動画ファイルとか、mp3 ファイルとか投げ込んでも問題ありません)

```
# マウントポイント作って ifuse でマウントする。
$ mkdir document2
$ ifuse --appid com.savysoda.documents2Free `pwd`/document2
$ cd document2
# document2 Free のストレージが見える
$ ls -l
Inbox/ mraid.js
$ mkdir 東京 Debian
$ cp /home/yours/doc/monthly-report/debianmeetingresume201403.pdf 東京 Debian/
$ cd ..
# unmount は fusermount コマンドを使う。
$ fusermount -u `pwd`/document2
```

Step 10. iphone5 側で Document 2 Free を立ち上げると、東京 Debian というフォルダが出来ており、さらにその下に pdf ファイルが見えます。こちらをタップすると、本資料を iphone5 上で読むことが出来ます。

6.4 仕組み

東京エリア Debian 勉強会に集まるような人にとっては、ただ繋ぐだけではおもしろくないと思いますので、仕組みについて説明します。

6.4.1 iphone アプリのストレージの約束事

iphone アプリは、apple 社が定めるいろいろな約束ごとに基づいて設計されています。

今回マウントした iphone アプリのストレージに関する約束ごとは、公開情報となっている Apple Developer サイト掲載の「ファイルシステム プログラミングガイド」[2] に詳細があります。この中で、知っておくべき事として、

- iphone アプリは各々割り当てられたサンドボックス内のリソースでしか動作を許されていません。そのため、例えば A アプリから B アプリのストレージ領域をアクセスすることがそもそも出来ません。
- ディレクトリの名前と用途が統一的に決められています。(例：Documents/,Library/,tmp/)

となっています (図 5 参照。) 今回マウントに利用した ifuse コマンドですが、このコマンドは、通常はオプション–appid で指定した ipone アプリの持つ Documents ディレクトリをマウントする能力があります。(注：所謂 jail break した iphone 端末は除きます)

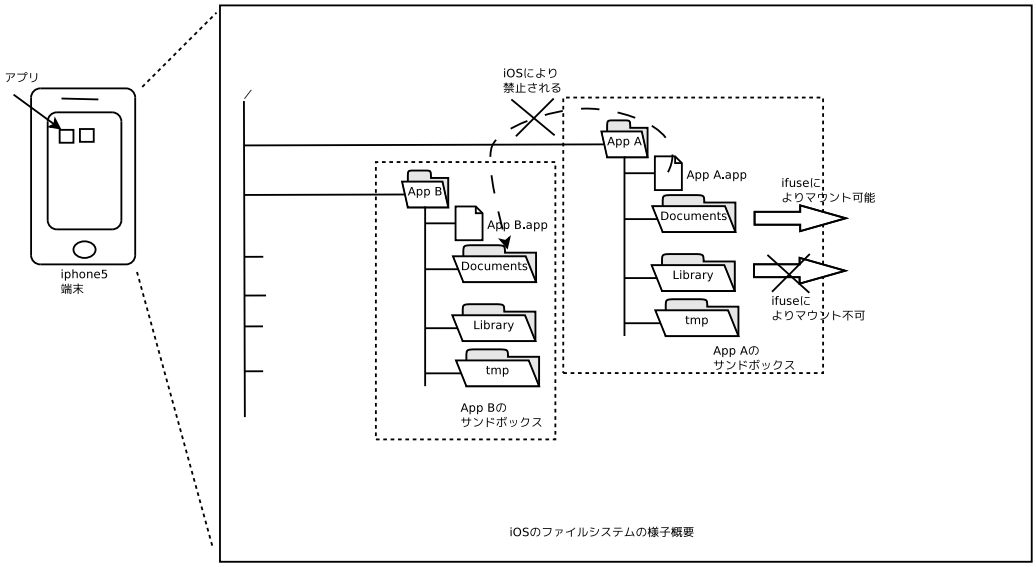


図 5　iphone アプリのストレージの様子

6.4.2　Debian との通信の仕組み

今回紹介の件について iphone5 と Debian との通信の仕組みを図 6 に示します。

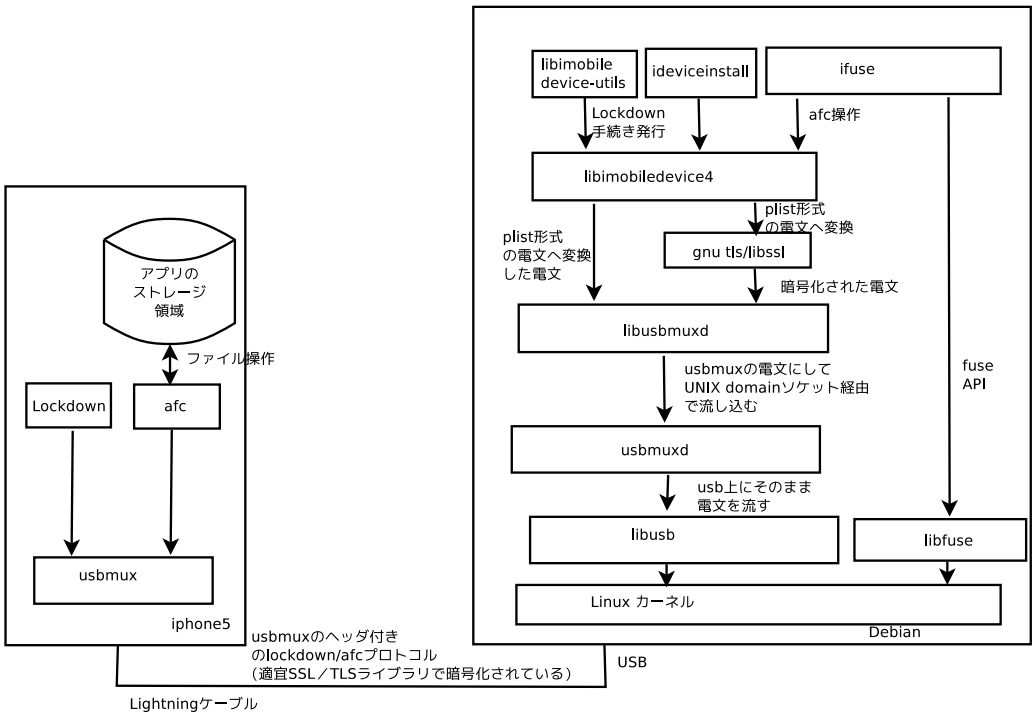


図 6　iphone5 と Debian の通信の仕組み

基本的に、Lightning-USB ケーブル上には、usbmux のパケットが流れます。　さらにペイロードとして、lockdown プロトコルと言われるプロトコル、afc プロトコルが適宜 SSL/TLS ライブラリで暗号化されて乗っています。

プロトコルについて階層化して図示すると図 7 に示します。

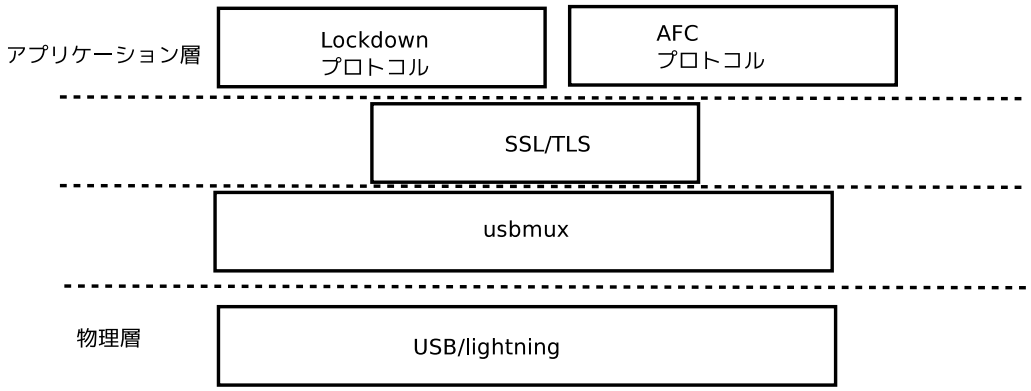


図 7　iphone5 との通信プロトコル

また、各々のプロトコルについての説明を表に示します。

項番	プロトコル名	概要
1	usbmux	lightning/usb に流れているプロトコル
2	Lockdown	iphone5 との通信認証、iphone5 の Lightning 端子側から利用出来るサービスのやりとりを担当。plist 形式の電文を usbmux に載せ、iphone5 とやりとりを行う。
3	afc	Apple File Connection のためのプロトコル。ファイルシステム操作が出来る。

表 6　プロトコル名称と概要

また、plist 形式とは、property list の略で、バイナリ形式の電文と、XML 形式の電文の 2 種類があるようです。なお、電文のやりとりの詳細、電文の説明については、文献 [3],[4], [5] に詳細があります。

6.5　終わりに

今回、Debian に iphone5 を繋ぎ、iTune によらないデータ転送について紹介しました。また、実現する技術についても紹介しました。こちらにより、不自由なスマートフォンを少しでも自由に使う事が出来れば幸いです。

参考文献

[1] GIGAZINE,「Apple が過去最高の売上を発表、日本では iPhone が 69% のシェアを獲得」, http://gigazine.net/news/20140128-apple-report-fy14-q1/

[2] Apple Developer サイト,「ファイルシステム プログラミングガイド」, https://developer.apple.com/jp/devcenter/ios/library/japanese.html

[3] theiphonewiki,“Usbmux”, http://theiphonewiki.com/wiki/Usbmux

[4] theiphonewiki,“afc”, http://theiphonewiki.com/wiki/AFC

[5] GOTO:Hack,“Hacking apple accessories to pown iDevices”, Mathieu RENARD, http://2013.hackitoergosum.org/presentations/Day3-04.Hacking%20apple%20accessories%20to%20pown%20iDevices%20%E2%80%93%20Wake%20up%20Neo!%20Your%20phone%20got%20pwnd%20!%20by%20Mathieu%20GoToHack%20RENARD.pdf

7 自宅サーバに KVM を導入してみよう

川江 浩

7.1 はじめに

現在、「クラウド」などで使われている「仮想化技術」ですが、Debian に代表される distribution のおかけで、パーソナルベースでも手軽に使えます。

そこで、仮想化技術の KVM を使ってネット関連のサーバ構築や市販の OS を install しましたので、システムを構築する際の注意点をレポートします。

また、以下の仕様はパーソナルベースでの運用を前提に構築したものです。仕様を試そうとするときは、必ずデータ等のバックアップをとって自己責任で行ってください。より詳しく知りたい方は専門書を参照してください。

7.2 仮想化技術

当初の仮想化は UNIX が動作するサーバや PC アーキテクチャ上で、スーパーバイザ（OS）がハードウェアを管理する形式でした。2000 年代になると IA（Intel Architecture）の処理能力が向上し、IA サーバの仮想マシンモニタが登場しました。

その中の Xen は仮想マシンソフトウェアの一つで、OS より 1 つの下の階層でハイパーバイザというプログラムを管理 OS が動かし、仮想化を実現します。

これに対して KVM（Kernel-based Virtual Machine）も同じくハイパーバイザですが、CPU の仮想化支援機能を利用して、機械全体をエミュレーションするシステムエミュレーションの QEMU を高速化し、仮想マシンの OS をアプリケーションとして動かします。

7.2.1 仮想化の実行モデル

仮想化の実行モデルとして、準仮想化と完全仮想化の 2 つがあります。

- 準仮想化 (ParaVirtualization)
  準仮想化はハードウェアをエミュレートする代わりに、仮想マシン用のハードウェアを使用します。このハードウェアは操作をするためにハイパーバイザコールを呼び出します。ハイパーバイザコールは仮想マシン環境に対応し、ゲスト OS は仮想ハードウェア用に修正する必要があります。
- 完全仮想化 (FullVirtualization)
  完全仮想化機能には Bibary Translation という手法や、CPU の仮想化支援機能を利用した手法があり、デフォルトの OS をそのままで動作させることができます。

今回は、qemu-kvm を取り上げましたので以下、仮想化支援を前提にします。

7.2.2 仮想化支援機能

CPU で仮想化支援機能は、Intel-VT や AMD-V などで、カーネルモードとユーザモード以外にもう一つゲストモードを追加しています。使っているパソコンのハードウェアが仮想化支援機能を利用できるかは、Intel VT-x/d、AMD-V をサポートしているかによります。

Intel では vmx を

```
$ grep vmx /proc/cpuinfo
flags   : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
 pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm
 constant_tsc arch_perfmon pebs bts rep_good nopl aperfmperf pni dtes64
 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 xsave
 lahf_lm dtherm tpr_shadow vnmi flexprioriy
```

AMD では svm を確認します。

```
$ grep svm /proc/cpuinfo
flags   : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
 pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt
 pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc extd_apicid
 aperfmperf pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 popcnt
 aes xsave avx f16c lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
 misalignsse 3dnowprefetch osvw ibs xop skinit wdt lwp fma4 nodeid_msr
 tbm topoext perfctr_core arat cpb hw_pstate npt lbrv svm_lock nrip_save
 tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold
```

7.3 KVM の導入

次に、qemu-kvm を install します。

```
# apt-get install qemu-kvm
# apt-get install libvirt-bin
```

同時に、GUI の virt-manager を使った方が仮想マシンの作成、実行、管理が簡単なので、install します。

```
# apt-get install virt-manager
```

install が終ったら、PC を再起動して、libvirt コマンドを実行するユーザを libvirt グループのメンバーにしておきます。

```
# adduser <user> libvirt
```

また、qemu-kvm 自体はカネールの機能で、package で導入する場合はカーネルモジュールとして動作します。

Intel VT の場合

```
$ lsmod | grep kvm
kvm_intel             121968  0
kvm                   287749  1 kvm_intel
```

AMD-V の場合

```
$ lsmod | grep kvm
kvm_amd                47218  10
kvm                   287749  1 kvm_amd
```

7.3.1 qemu-kvm の仮想ネットワーク

次に、仮想ネットワークの構成ですが、qemu-kvm の仮想ネットワークは default で nat を使うようになっています。ただし、ネット関連のサーバを構築する場合、サーバを「外部」に公開する必要があります。そこで virt-manager を使って、仮想ネットワークを接続する仮想 bridge を作成します。この場合のネットワークインターフェイスは QEMU でエミュレートされ、TAP デバイスを作成して単純に入出力を接続する「tap」を作ります。この tap は擬似的な Ethrenet デバイスで Linux カーネルの機能です。また、この仮想的な Ethrenet デバイスを、同じく Linux のカネールの機能である仮想 bridge で接続するこで、VM は実デバイスやほかの VM と接続することができます。

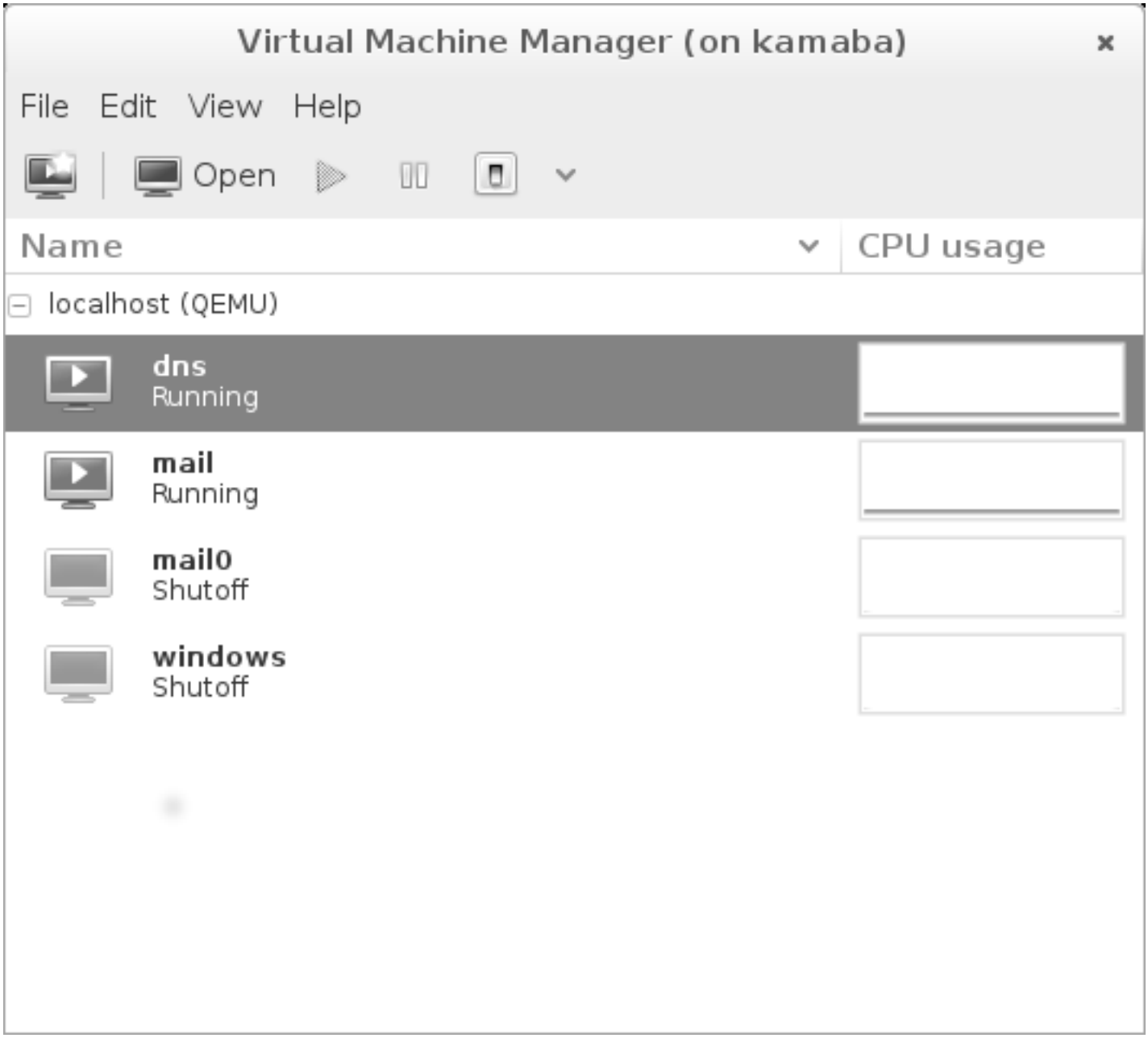


図 8　virt-manager の図

7.3.2 仮想 bridge の作成

次に、仮想 bridge を作成します。これは、ソフトウェアで 802.1d Ethernet ブリッジを実装しています。手順としては、

1. br0 を作成
2. eth0 を初期化、br0 に接続
3. VM の仮想 NIC に対応するインターフェイスを br0 に接続

具体的には、virt-manager を立ち上げ「Connection Details」で、「NetworkInterfaces」の「Configure network interface」で「Bridge」を選択し、仮想 bridge にするデバイスを選択して IP アドレス（IPv4）をふります。

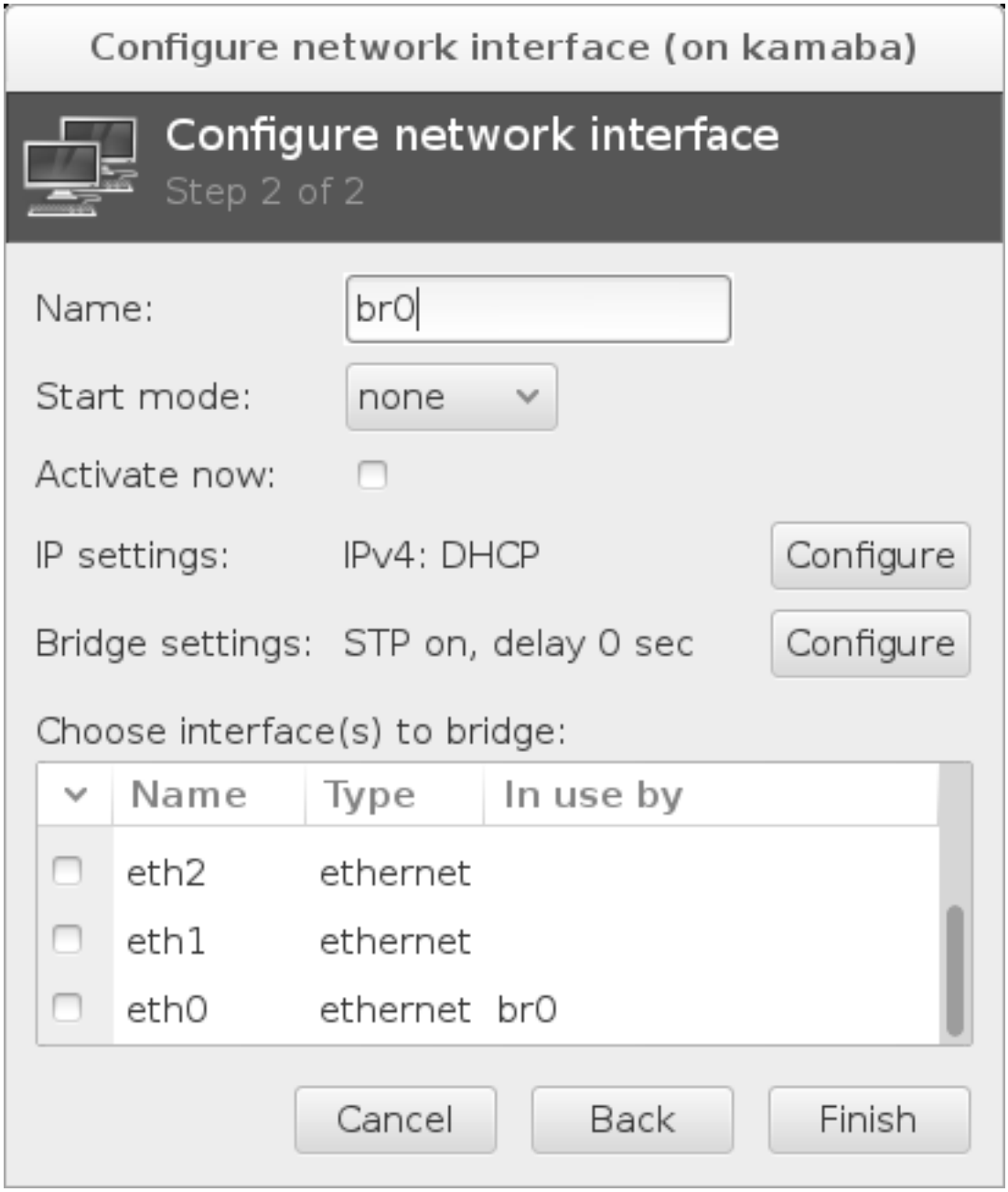


図 9　bridge 設定の詳細の図

後は、VM を作成するときにネットワークで「bridge」を選択すればいいだけです。イメージとしては以下の「図」のようになります。

ここで eth0 は「bridge」として外部につながります。また、セキュリティの関係から eth1 は別「segment」にし、eth0 から通信を遮断します。この eth1 では SSH などを使用し、virt-manager で各 VM を管理します。そして eth2 は eht0 と同じ「segment」にして、後述の Spice 専用とします。

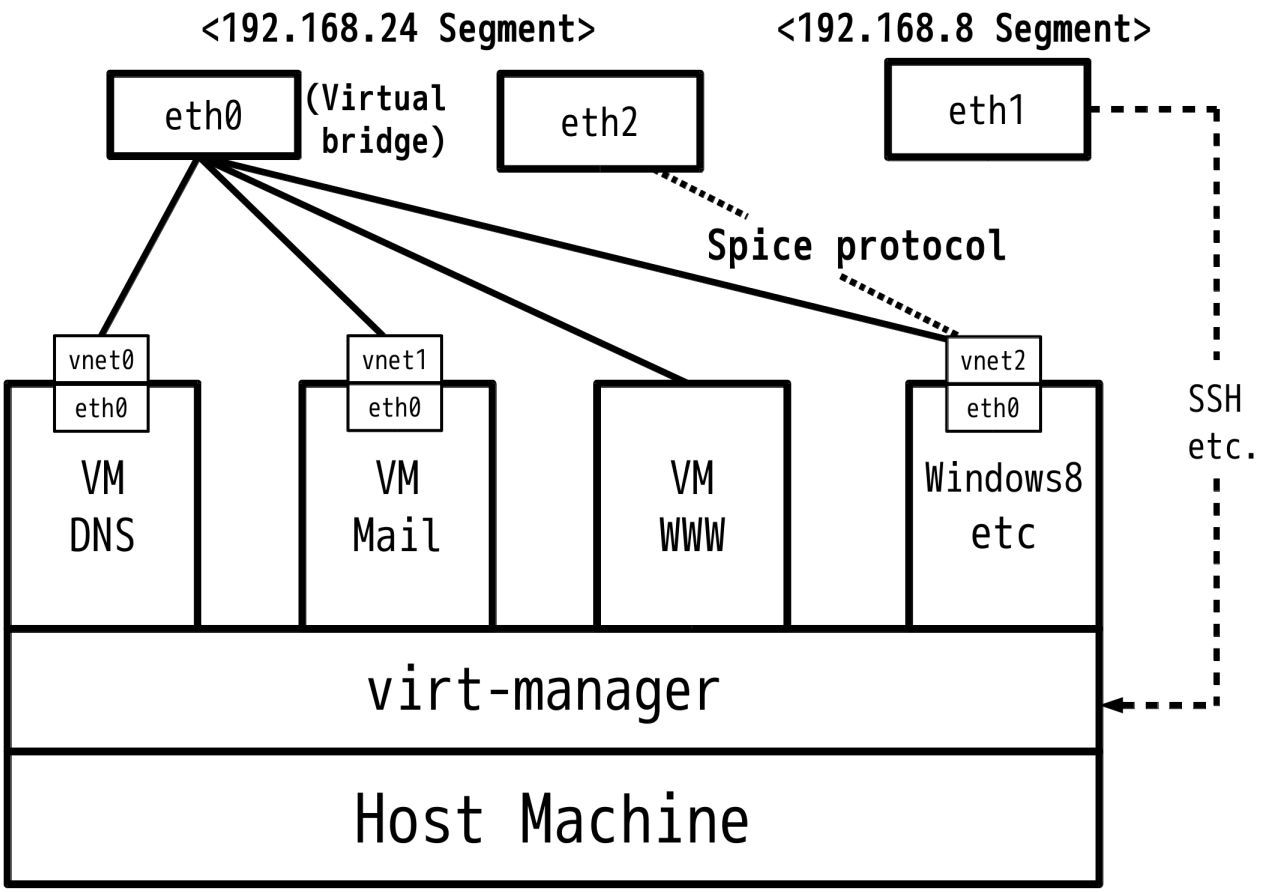


図 10　ネットワーク構成の図

7.3.3　Virtual Machine の作成

次に、virt-manager で VM を作成します。virt-manager は GUI ツールのウィザードで、VM の作成時には、物理マシン用のインストールディスク（市販 OS）や ISO イメージを使い VM を作成します。また、実マシンのリソース内であれば容易に VM の「拡張」ができます。

7.4 各種インターネット関連のサーバの作成

VM の作成ができたら、各サーバを作成します。まず、DNS の作成には bind9 パッケージをインストールします。

```
# apt-get install bind9
# apt-get install bind9utils
```

DNS の設定ですが NTT の回線の関係上、１個の「グローバル」IP アドレスしか使えなかったので、複数のローカル IP アドレスで各サーバ（クライアント）に IP アドレスを振ります。そこで、acl（Access Control List）で IP アドレスとネットワークを指定し、acl にマッチするクライアントと、その他のクライアントごとに別々ゾーンを提供します。これを利用し、1 台のサーバで内部用（acl にマッチする）と外部用（acl にマッチしない）の DNS を構築します。具体的には、

1. acl で、アドレスマッチリストを設定。
2. クライアント（各サーバ）を IP によって指定し、クライアント毎の振る舞いを作成。
3. view ステートメントで、内部クライアント、外部クライアントに別々の zone ファイルを参照させる。

Mailserver には、Postfix（MTA Mail Transfer Agent）、Dovecot（MDA Mail Delivery Agent）を使います。

```
# apt-get install postfix
# apt-get install dovecot
```

まず、Postfix については、sasl の認証機構を使ってユーザの認証（SMPT-AUTH）を行い、ユーザ認証で用いるパスワード（平文）を TLS で暗号化します。そして、メール送信のために認証機能のついたサブミッションポートを使用するための設定をします。

devecot は IMAP（Internet Message Access Protocol）プロトコルを使いメールサーバのメールにアクセス、操作、オフラインとオンラインの双方で利用できます。また、Mail サーバの構成については、

1. 受信したメールを Mail ディレクトリで扱うように設定
2. imapcopy を使って、旧メールサーバよりメールデータの転送
3. ウィルス対策には、clamtk をインストール

7.5 サーバの運用・管理

サーバーの運用・管理は、リモート操作、バックアップ、バッチ処理の実行、トラブルの予防・発見のための稼働監視などが必要ですが、個人で運用する自宅サーバなので、VM のクローンと import、SPICE を取り上げます。

7.5.1 VM の clone、import

バックアップは、VM である各サーバのディスクイメージをベースに、clone を作ります。この clone（ディスクイメージ）はベースとなった VM とまったく同じ「構成」ですが、virt-manager を使って MAC アドレスの変更ができます。また、イメージを KVM が走っている別のマシンに import できます。また、各種サーバの構築の際は「ベースの VM」を作成し、クローンを作って『カタマイズ』するなどの使い方ができます。

7.5.2 SPICE

リモート操作には SPICE を使います。SPICE（Simple Protocol for Independent Computing Environment）は、仮想化環境上に構築したデスクトップ環境にリモートから接続するという仮想デスクトップ環境（VDI）です。特徴は、オープンソースの画面転送プロトコルで、高画質なビデオ転送と高音質な双方向音声転送ができることです。

次に、spice クライアントをインストールし、ターミナルから接続します。

```
# apt-get install spice-client
```

続いて、qemu-kvm の VM に接続します。

```
$ spicec -h noren.kinsen.gr.jp -p 590X -w *********
```

「-h」はホスト名、「-p」はポート番号、「-w」はログインするためのパスワードです。

7.6 まとめ

まとめとして、自分の自宅サーバはこの間、2 年ほど「Xen」で運用していました。Xen は qemu-kvm と同じハイパーバイザなのですが、仮想化のモデルが準仮想化なので、サーバに多くのリソースは必要ありません（メールサーバ、DNS ともども 384MB で運用しました)。ただ、リソースが十分に用意できるのであれば、OS を default で使える点や、virt-manager などのツール、「高速」「高機能」の SPICE が使用できるなどの利点が qemu-kvm にはあります。

今後の予定としては、WWW サーバでは「HTML5」を使ったサイトの構築や、IPv6 への対応（ただし、パーソナルレベルで Global Routing Prefix が /48-/64bit 内であることが前提)、さらなる『異種』の OS の VM 化をしたいと思っています。

8 Golang で書かれたツールを Debian パッケージにする

前田 耕平

8.1 はじめに

あることをやるのに Serf[*4]というツールが良いのでは、と同僚に助言をもらったので、Serf でやりたいことができないか検証をすることにしました。ですが、そもそも Debian パッケージには無いのでまずそこからでしょ、ということで手元で Debian パッケージにしました。[*5]

Serf を Debian パッケージにするにあたり、行ったことや、Golang のツールを Debian パッケージにする上で必要な知識、手順をまとめました。今回の勉強会では、Debian パッケージを初めて作成する人の割合が比較的多かったので、内容的に少し冗長になっています。

8.2 Debian での Golang の環境構築

8.2.1 前提条件

使用するディストリビューションは、Debian GNU/Linux Sid を前提とします。手元に Sid の環境がない場合は、仮想マシンなどで用意してください。

8.2.2 Debian パッケージのインストール

Debian では、“golang” というメタパッケージが用意されています。これをインストールすると次のパッケージがインストールされます。[*7]

- golang-doc
  `http://golang.org` のドキュメント。`godoc --http=:6060` を実行し、`http://localhost:6060/` で閲覧可
- golang-go
  Golang のアセンブラ、コンパイラー、リンカーなどのツールチェイン
- golang-go-linux-amd64
  Linux システム、amd64 アーキテクチャ向けの標準ライブラリ。system: darwin,freebsd,linux,netbsd,windows、arch: amd64,386,arm
- golang-go.tools
  Golang 用の補助ツール。前述の godoc コマンドも含まれる

[*4] `http://www.serfdom.io/`
[*5] Serf 自体のお話を聞きたいと思った方は残念。Upstream[*6]のドキュメントや日本語の Serf を使ったオーケストレーションの資料を見ただけで、Serf 自体の検証はまだ行っていないので、Serf そのものについては何も語れません。
[*7] Linux Kernel の amd64 アーキテクチャの場合

- golang-src
  golang パッケージのソースコード。godoc コマンドなどで使われる
- libjs-jquery
  jQuery。golang-go.tools に依存。godoc コマンドで実行する golang-doc のドキュメント用
- javascript-common
  JavaScript ライブラリパッケージのサポートパッケージ。libjs-jquery に依存

8.3 Golang のコンパイル方法

標準ライブラリのみで書かれたコードをコンパイルする場合には、`go build` コマンドのみでできます。例えば、おなじみに Hello world のコード (hello.go) を用意します。

```
package main

import "fmt"

func main() {
  fmt.Println("Hello world.")
}
```

コンパイルします。

```
$ go build hello.go
```

バイナリの実行は`./hello` です。

```
$ ./hello
Hello world.
```

`file` コマンドで見ると、静的リンクされていることが分かります。

```
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
```

なお、`go run` コマンドを使うと、コンパイルしなくても実行可能です。

```
$ rm hello
$ go run hello.go
Hello world.
$ ls hello
ls: cannot access hello: No such file or directory
```

8.3.1 標準ライブラリ以外に依存する Golang パッケージのコンパイル

まず、依存するパッケージの配置場所を環境変数 `GOPATH` で指定する必要があります。

システムグローバルの設定は、Debian では、`/usr/share/gocode` になります。既に Debian パッケージになっている Golang のライブラリは、このディレクトリの下にある、src ディレクトリ以下にインストールされます。

システムグローバルで設定されているパッケージを使う場合には、

```
$ export GOPATH=/usr/share/gocode
```

GOPATH は絶対パスで指定する必要があります。コンパイルするコードと同じカレントディレクトリに存在するなら、

```
$ export GOPATH=$(pwd)
```

と実行した上で、実際のパッケージの配置は、`$(pwd)/src/example.org/mkouhei/hoge/`のようなディレクトリ構成になっている必要があります。先ほどの hello.go にこのパッケージを import する場合、

```
import (
    "fmt"
    "example.org/mkouhei/hoge"
)
```

のようになります。

なお、標準ライブラリは、Debian では、/usr/lib/go 以下にインストールされています。標準ライブラリライブラリは、GOPATH で指定する必要はなく、GOROOT という環境変数で、/usr/lib/go が設定されています。

GOPATH が設定されていれば、パッケージのダウンロードとインストールは go get コマンドで行えます。[*8]

GOPATH で指定したディレクトリ以下に、下記のような形で自動的にインストールされます。

```
$ go get example.org/mkouhei/hoge
$ tree
.
|-- pkg
|   `-- linux_amd64
|       `-- example.org
|           `-- mkouhei
|               `-- hoge.a
`-- src
    `-- example.org
        `-- mkouhei
            `-- hoge
                |-- LICENSE
                |-- README.rst
                `-- hoge.go
```

GOPATH を設定し、import できれば、go build は依存関係を解決してコンパイルできます。バイナリのみを Debian パッケージにする際に、GOPATH が重要です。

8.4 Golang で書かれたソフトウェアの Debian パッケージ作成方法

Golang で書かれたソフトウェアの Debian パッケージ作成方法について説明します。大きく次の 2 パターンがあります。

- バイナリのみ
- ライブラリのみ（もしくはライブラリとバイナリ）

前者ではコンパイルが必要です。Golang の場合、依存するライブラリを全て静的にリンクするため、バイナリの実行そのものには依存関係はありません。ですので、デーモン化するケースでなければ、Depends に記述するパッケージが基本必要ありません。[*9]

後者では、基本的にはコンパイルそのものは必要ありません。しかし、前述の/usr/share/gocode 下にソースコードをインストールする必要があるので、dh-golang パッケージを Build-Depends 記述し、debian/rules で、dh コマンドに--with=golang オプション渡してやる必要があります。

また、バイナリ、ライブラリに問わず、依存する Golang のパッケージを全て Build-Depends に記述する必要があります。[*10]また、Golang では go test でテストを実行したり、go build や Makefile でコンパイルする場合、依存する Golang パッケージを go get で GOPATH で指定したディレクトリにダウンロードおよびインストールされるのですが、Debian パッケージのビルドでは、go get を使わないようにしなくてはいけません。

8.4.1 Debian パッケージ作成用の環境設定

まず dh-golang パッケージをインストールします。Debian パッケージ自体を作ったことが無く、一から環境を整える、という人は、次のコマンドを実行します。

[*8] http://golang.org/cmd/go/#hdr-Download_and_install_packages_and_dependencies

[*9] デーモン化して OS 起動時に自動起動する場合、ログのローテーションで logrotate に依存したり、専用ユーザを作って実行させるのに adduser が必要です。

[*10] これは Golang に限った話ではありませんね。

```
$ sudo apt-get devscripts debhelper fakeroot dh-golang cowbuilder piuparts git git-buildpackage
```

git および git-buildpackage[*11]は必須ではありません。しかし、Golang で書かれたのツールの多くは Git リポジトリで管理されています。また、昔ながらのバージョニングがされないケースがほとんどです。そのため、tarball 自体の提供もされないものが多いので、Git が通常必要になります。

また Golang の Debian パッケージのメンテナンスチーム (the pkg-go team[*12]) では、git-buildpackage でソースパッケージを管理することになっています。[*13]

なお、前述の `go get` コマンドでは、リポジトリから入手して、そのまま `GOPATH` で設定したディレクトリの src ディレクトリ以下に展開されてしまうので、バージョンやコミットが分からないので使わない方がよいでしょう。なお、Build-Depends での依存関係の解決の際にも、Debian パッケージングポリシー上、`go get` コマンドは使えません。

.bashrc に下記のような変数を設定します。

```
export DEBFULLNAME='Kouhei Maeda'
export DEBEMAIL=mkouhei@palmtb.net
```

pbuilder, cowbuilder の chroot イメージを作成します。[*14]

```
$ sudo pbuilder --create
$ cowbuilder --create
```

pbuilder の方は、/var/cache/pbuilder/base.tgz が、cowbuilder の方は/var/cache/pbuilder/base.cow が作成されます。

8.4.2 ライブラリのパッケージ作成

まず Upstream の tarball もしくは SCM のリポジトリを入手します。前述の通り、Git で管理されているケースを想定します。

Serf のビルドの依存関係で (間接的に) 必要になった `https://github.com/huin/gobinarytest` を例にします。`git clone` したら、ブランチとタグ、ログを確認します。

```
$ git clone https://github.com/huin/gobinarytest.git
$ cd gobinarytest
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
$ git tag -l
$ git log | head -5
commit de322f729af68c9e5ecde2bff780d105f0fbe745
Author: Huin <greatred@gmail.com>
Date:   Tue Feb 11 22:50:34 2014 +0000

    Add LICENSE
```

Golang で書かれたツールは、この通り、伝統的なバージョンニングがされないことが多いです。`go get` コマンドを使えば、依存関係のあるパッケージの最新のコミットを入手できることが影響しているのではないかと　思います。

pkg-go team のポリシーとして、こういうケースには、0.0 gitYYYYMMDD のようなバージョンをつける事になっています。[*15]

また、Debian パッケージを作成するときの命名規則もあります。Golang の場合、Perl の CPAN や、Python の PyPI のような公式に一元管理しているパッケージリポジトリが存在しないので、Git のリポジトリ名だけを使うと名前が競合する可能性があります。[*16]

[*11] git-buildpackage はソースパッケージ自体を Git で管理するためのツールです。
[*12] `http://pkg-go.alioth.debian.org/`
[*13] `http://pkg-go.alioth.debian.org/packaging.html#_packaging_in_git`
[*14] pbuilder は cowbuilder をインストールすると自動的にインストールされます。
[*15] `http://pkg-go.alioth.debian.org/packaging.html#_version_numbers`
[*16] Serf をパッケージ化する際にも実際にありました。

そこで、ライブラリパッケージを作成する場合、github.com/huin/gobinarytest の場合には、golang-huin-gobinarytest-dev というパッケージ名にします。[*17]

では、まず `git archive` コマンドを使って tarball を作ります。

```
$ git archive --prefix=golang-huin-gobinarytest/ HEAD | gzip > ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
```

次に Debian パッケージ用の Git リポジトリを作成し、tarball をインポートします。

```
$ cd -
$ mkdir golang-huin-gobinarytest
$ cd golang-huin-gobinarytest
$ git init
$ git import-orig ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
What will be the source package name? [golang-huin-gobinarytest]
What is the upstream version? [0.0~git20140211]
gbp:info: Importing '../golang-huin-gobinarytest-0.0~git20140211.tar.gz' to branch 'master'...
gbp:info: Source package is golang-huin-binarytest
gbp:info: Upstream version is 0.0~git20140211
gbp:info: Successfully imported version 0.0~git20140211 of ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
```

次に `dh_make` コマンドで debian ディレクトリを生成します。このパッケージ自体は 2 条項 BSD ライセンスなので、`--copyright` オプションで指定します。作成後、不要なテンプレートは削除します。ライブラリパッケージですが、

```
$ dh_make -l --copyright bsd -p golang-huin-gobinarytest_0.0~git20140211 -f ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
$ cd debian
$ ls
README.Debian  golang-huin-gobinarytest-dev.dirs     init.d.ex        preinst.ex
README.source  golang-huin-gobinarytest-dev.install  manpage.1.ex     prerm.ex
changelog      golang-huin-gobinarytest.cron.d.ex    manpage.sgml.ex  rules
compat         golang-huin-gobinarytest.default.ex   manpage.xml.ex   shlibs.local.ex
control        golang-huin-gobinarytest.doc-base.EX  menu.ex          source
copyright      golang-huin-gobinarytest1.dirs        postinst.ex      watch.ex
docs           golang-huin-gobinarytest1.install     postrm.ex
$ rm README* golang-huin-gobinarytest* init.d.ex manpage.* menu.ex post* pre* shlibs.local.ex watch.ex
$ ls
changelog  compat  control  copyright  docs  rules  source
```

下記のように変更します。

debian/control

Golang のパッケージには、Build-Depends に golang-go と、また Golang のライブラリパッケージは Depends にも golang-go と、また Build-Depends に dh-golang を記述します。ライブラリの golang-huin-gobinary-test-dev パッケージは、ソースコードなので、Architecture は all にします。

```
Source: golang-huin-gobinarytest
Section: devel
Priority: optional
Maintainer: Kouhei Maeda <mkouhei@palmtb.net>
Build-Depends: debhelper (>= 9.0.0), dh-golang, golang-go
Standards-Version: 3.9.5
Homepage: https://github.com/huin/gobinarytest

Package: golang-huin-gobinarytest-dev
Architecture: all
Depends: golang-go
Description: Helper code for unit testing binary encoding code
 The gobinarytest package is used in testing serialization and
 deserialization. It is particularly useful when some sub-sequences of bytes
 can acceptably be written in any order (e.g if they are generated from
 representations that do not provide order guarantees like maps).
```

debian/rules

Golang 用のポイントとしては、`dh` コマンドに、`--buildsystem=golang` オプションと、`--with=golang` オプションを指定することです。

ライブラリ用には、先ほどの `GOPATH` の下で展開されるパッケージのネームスペースを環境変数 `DH_GOPKG` で指定します。このネームスペースは、通常公開しているリポジトリの URL がベースになりますが、場合によっては

[*17] `http://pkg-go.alioth.debian.org/packaging.html#_naming_conventions_2`

実際に公開している URL とは別のリダイレクト元の URL を指定することこともあります。upstream のドキュメントやソースコードを確認してみてください。また、override_dh_auto_install を定義し、dh_auto_install -O-buildsystem=golang で上書きします。これにより、このパッケージをビルド後、golang-huin-gobinarytest-dev パッケージをインストールすると、前述の/usr/share/gocode の下にソースコードがインストールされます。

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

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

export DH_GOPKG := github.com/huin/gobinarytest

%:
	dh $@ --buildsystem=golang --with=golang

override_dh_auto_install:
	dh_auto_install -O-buildsystem=golang
```

この例では、他の Golang パッケージに依存していないので使っていませんが、他のパッケージに依存する場合は、

```
export GOPATH := $(CURDIR)_build:/usr/share/gocode
```

のような設定が必要になります。

debian/copyright

```
ormat: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: gobinarytest
Source: https://github.com/huin/gobinarytest

Files: *
Copyright: 2013, John Beisley <greatred@gmail.com>
License: BSD-2-Clause

Files: debian/*
Copyright: 2014 Kouhei Maeda <mkouhei@palmtb.net>
License: BSD-2-Clause

License: BSD-2-Clause
 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:
 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.
 .
 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE HOLDERS OR
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

debian/changelog

dch -r コマンドを実行します。

```
golang-huin-gobinarytest (0.0~git20140211-1) unstable; urgency=low

  * Initial release (Closes: #nnnn)

 -- Kouhei Maeda <mkouhei@palmtb.net>  Wed, 12 Feb 2014 14:47:33 +0900
```

あとは golang-huin-gobuildtest ディレクトリの下で、git buildpackage コマンドを実行すると、予め cowbuilder --create コマンドで作成しておいた/var/cache/pbuilder/base.cow の chroot イメージを使って、パッケージがビルドされます。

8.4.3 バイナリのパッケージ作成

大まかな流れはライブラリの場合と同じです。異なるのは、

- パッケージの命名規則
  golang- という prefix をつける必要はない。[*18]
- debian/control の Build-Depends に dh-golang が不要[*19]
- debian/control のバイナリ用のパッケージの Architecure が all ではなく、any
- debian/rules で、`dh` コマンドに`--with=golang` オプションは不要 (dh-golang 用のオプションのため)

というあたりです。
Serf[*20]をパッケージにする場合の設定例は下記です。

8.4.4 debian/control

Build-Depends の中で、ライブラリの場合と違うのは txt2man と dh-systemd です。実行可能なバイナリファイルは、Debian パッケージのポリシー上、man マニュアルを用意する必要があります。しかし、serf はコマンドラインヘルプは充実しているものの、man マニュアルはありません。なので、txt2man を使って自動生成しました。
dh-systemd は、systemd 用の補助ツールです。

```
Source: golang-serf
Section: web
Priority: optional
Maintainer: Kouhei Maeda <mkouhei@palmtb.net>
Build-Depends: debhelper (>= 9.0.0),
        golang-go (>= 2:1.2),
        golang-hashicorp-memberlist-dev,
        golang-mitchellh-cli-dev,
        golang-hashicorp-logutils-dev,
        golang-mitchellh-mapstructure-dev,
        golang-ugorji-go-dev,
        golang-armon-mdns-dev (>= 0.0~git20140225~8be7e3a),
        golang-ryanuber-columnize-dev,
        dh-systemd,
        txt2man
Standards-Version: 3.9.5
Homepage: http://www.serfdom.io/

Package: serf
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}, adduser
Description: Service orchestration and management tool
 Serf is a service discovery and orchestration tool that is decentralized,
 highly available, and fault tolerant. Serf runs on every major platform:
 Linux, Mac OS X, and Windows. It is extremely lightweight: it uses 5 to 10 MB
 of resident memory and primarily communicates using infrequent UDP messages.
```

debian/rules

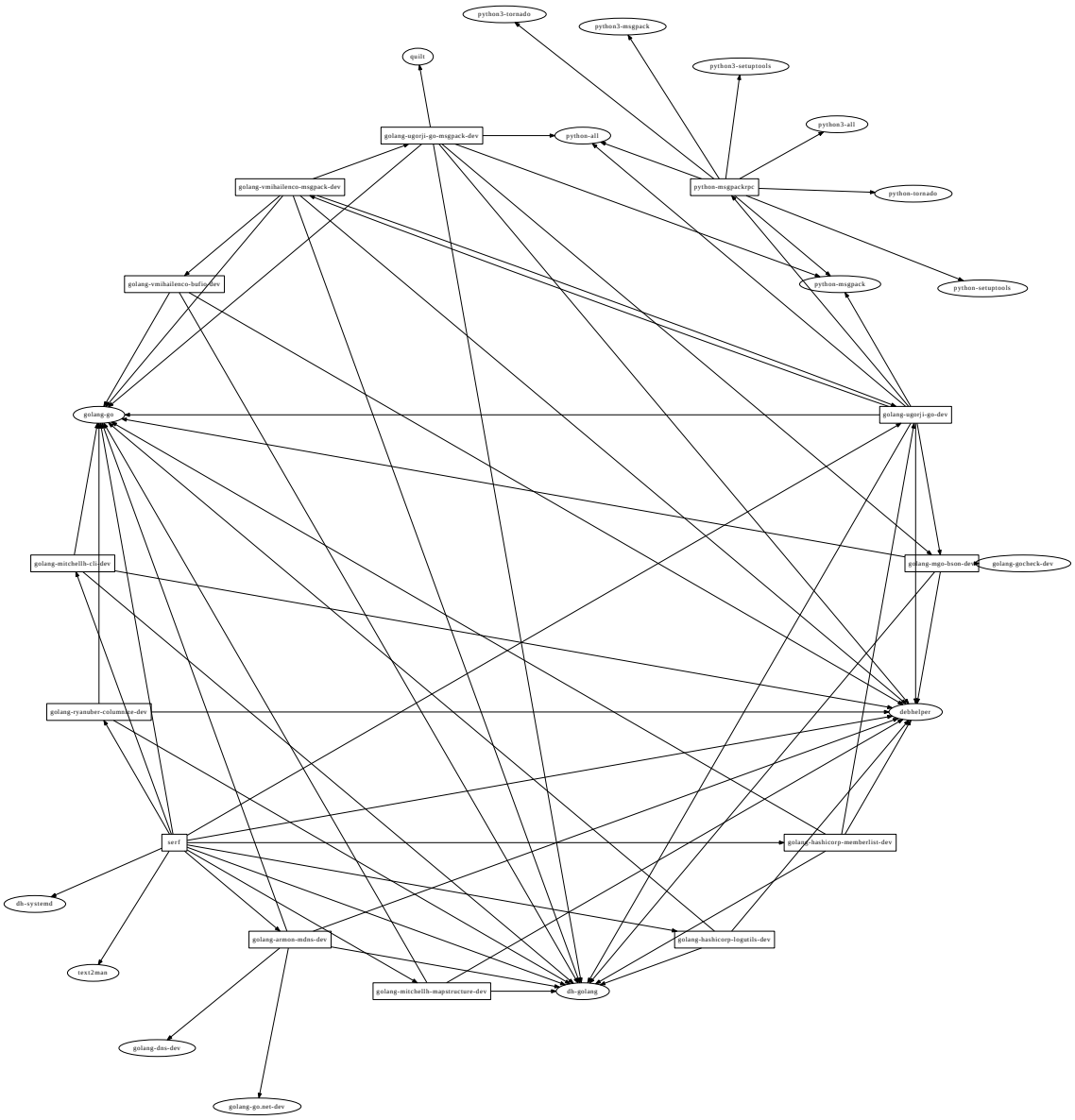
Serf には、Makefile がついているのですが、ビルドに前述の `go get` コマンドを使って、依存するパッケージをカレントディレクトリにダウンロード＆インストールしています。Debian パッケージの作成のポリシー上、ビルド中にリモートホスト上のリソースを取得してはいけないので、upstream の Makefile をそのまま利用できません。そのため、カレントディレクトリと、システムグローバルに設定した GOPATH から、依存パッケージ及び、Serf 自身でインポートしているコードを使うようにしているのが、`override_dh_auto_build` の処理です。
また、前述の man マニュアルの生成は、`override_dh_installman` の処理で行っています。

[*18] Serf の場合は、ソースパッケージが serf となっている、HTTP client library(libserf1) があるので、バッティングするのでソースパッケージ名は golang-serf とするか、Docker のように serf.io とするのが良さそう。
[*19] ライブラリを含める場合は必要です。
[*20] `https://github.com/hashicorp/serf`

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

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

export DH_GOPKG := github.com/hashicorp/serf
GOPATH := ${CURDIR}/_build:/usr/share/gocode
BLDPATH := obj-$(shell dpkg-architecture -qDEB_BUILD_GNU_TYPE)
export LANG := C

export GOPATH

%:
        dh $@ --buildsystem=golang --with=systemd

override_dh_auto_build:
        mkdir -p ${CURDIR}/_build/src/${DH_GOPKG}
        cp -a ${CURDIR}/*.go ${CURDIR}/serf ${CURDIR}/client \
                ${CURDIR}/command ${CURDIR}/testutil ${CURDIR}/_build/src/${DH_GOPKG}/
        go build -v -o ${CURDIR}/_build/serf

override_dh_auto_test:
        cd $(CURDIR)/_build && \
        go list ./... | xargs -n1 go test && \
        $(CURDIR)/scripts/setup_test_subnet.sh && \
        go list ./... | INTEG_TESTS=yes xargs -n1 go test

override_dh_auto_install:
        install -d ${CURDIR}/debian/serf/usr/bin
        test -f ${CURDIR}/_build/serf && \
        install -m 0755 ${CURDIR}/_build/serf ${CURDIR}/debian/serf/usr/bin/

override_dh_installman:
        test -f ${CURDIR}/_build/serf && \
        ${CURDIR}/_build/serf -h 2>&1 | txt2man -t SERF -s 1 -v 'Serf Manual' > ${CURDIR}/debian/serf.1
        set -e && ${CURDIR}/_build/serf -h 2>&1 | grep -v version | grep -E '^\s+\w+' | while read line; do \
                subcmd=`echo $$line | awk '{print $$1}'`;\
                desc=`echo $$line | awk '{ $$1=""; print }'`;\
                ${CURDIR}/_build/serf $$subcmd -h 2>&1 | txt2man -t SERF-`echo $$subcmd | tr "[:lower:]" "[:upper:]"`
 -s 1 -v 'Serf Manual' > $(CURDIR)/debian/serf-$$subcmd.1; \
                sed -i "/Serf Manual/a .SH NAME\nserf-$$subcmd \- $$desc" $(CURDIR)/debian/serf-$$subcmd.1 ;\
        done
        sed -i '/Serf Manual/a .SH NAME\nserf \- Service orchestration and management tool' $(CURDIR)/debian/serf.1
        sed -i '/[Uu]sage:/i .SH SYNOPSIS' $(CURDIR)/debian/serf*.1
        sed -i '/Available commands are:/i .SH OPTIONS' $(CURDIR)/debian/serf*.1
        sed -i 's/^Options:/.SH OPTIONS/g' $(CURDIR)/debian/serf*.1
        sed -i 's/-/\\-/g' $(CURDIR)/debian/serf*.1
        dh_installman ${CURDIR}/debian/serf*.1

override_dh_auto_clean:
        dh_clean
        rm -rf ${CURDIR}/_build
        rm -rf ${CURDIR}/${BLDPATH}
        rm -f ${CURDIR}/debian/serf*.1

DEB_UPSTREAM_VERSION=$(shell dpkg-parsechangelog | sed -rne 's,^Version: ([^+]+).*,\1,p')
get-orig-source:
        uscan --noconf --force-download --rename --download-current-version --destdir=.
        rm -rf serf-$(DEB_UPSTREAM_VERSION)
        tar xf golang-serf_$(DEB_UPSTREAM_VERSION).orig.tar.gz
        rm -f golang-serf_$(DEB_UPSTREAM_VERSION).orig.tar.gz
        rm -rf serf-$(DEB_UPSTREAM_VERSION)/website
        GZIP=--best tar cz --owner root --group root --mode a+rX \
                -f ../golang-serf-$(DEB_UPSTREAM_VERSION)+dfsg.orig.tar.gz \
                serf-$(DEB_UPSTREAM_VERSION)
        rm -rf serf-$(DEB_UPSTREAM_VERSION)
```

8.5 Serf の依存するパッケージ

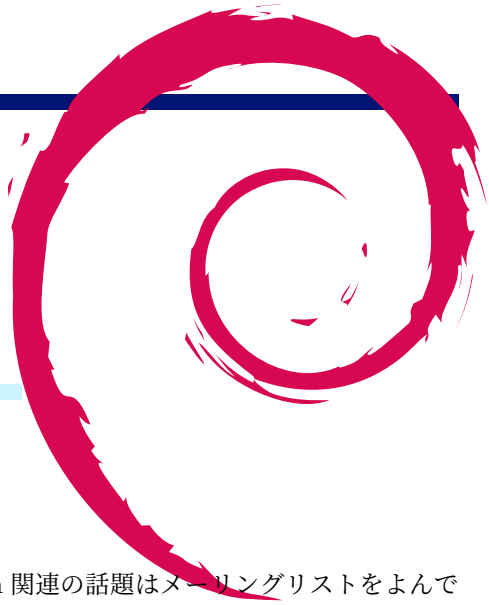
最後に、Serf が Build-Depends で必要となり、かつ、まだ Debian パッケージになっていないパッケージを図示しておきます。四角のノードがパッケージになっていないものです。



参考文献

[1] The Go Programming Language, “Command go”, `http://golang.org/cmd/go/`

[2] Alioth, “Debian Go Packaging”, `http://pkg-go.alioth.debian.org/packaging.html`



9 Debian Trivia Quiz

野島 貴英

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。

今回の出題範囲は `debian-devel-announce@lists.debian.org` や `debian-devel@lists.debian.org` に投稿された内容と Debian Project News からです。

問題 1. 2013/12/15 に出た Debian wheezy のアップデートのバージョンは？

☐ A 7.3
☐ B 7.2
☐ C 7.1

問題 2. Debian も参加している FOSS 貢献者に女性を増やそう運動の事をなんという？

☐ A Encourage Women in Linux
☐ B Outreach Program For Women
☐ C IT 戦士

問題 3. 2013/11/29 にて Debian の Technical Committee にメンバが増えました。ところで、2013/11/29 現在の Technical Comittee の chair man は誰でしょう?

☐ A Takahide Nojima
☐ B Lucas Nussbaum
☐ C Bdale Garbee

問題 4. PHP のメンテナチームを 3 つに分割する事が提案されました。分割されたグループの名前で間違っているのはどれ？

☐ A Debian PHP PECL Maintainers
☐ B Debian PHP PEAR Maintainers
☐ C Debian PHP DOCUMENT Maintainers

問題 5. Debian について、Debian Developer 以外の人でも貢献したを讃えましょうということで、作られたサイトは？

☐ A advocates.debian.org
☐ B contributors.debian.org
☐ C superstar.debian.org

問題 6. 2013/12 後半頃に s390x アーキテクチャのデフォルト C コンパイラとしての gcc のバージョンが変更されました。どのバージョンになったのでしょう？

☐ A 4.8
☐ B 4.7
☐ C 4.6

問題 7. 2014/1 に有名なデータベースがパッケージとして追加されました。何というデータベースでしょうか？

☐ A Maria DB
☐ B Percona DB
☐ C GDB

問題 8. 2014/2/1、次期 Debian のバージョンである Jessie にて、とあるアーキテクチャが取り除かれた事がアナウンスされました。それは、どのアーキテクチャ？

☐ A hurd-i386
☐ B s390x
☐ C ia64

問題 9. 2014/2 あたまにリリースされた stable 版の Debian のバージョンはいくつでしょう？
☐ A 7.3
☐ B 7.4
☐ C 7.5

問題 10. Debian の資産 (Asset の事です) を任せることのできる「信頼に足る組織 (Trusted Organization)」の定義が先日レビューされていました。信頼に足る組織の条件に当てはまらない組織はどれ
☐ A 公式 Debian 開発者が居ない組織
☐ B 素早い応答/対応ができる組織
☐ C Debian 社会憲章と対立しない組織

問題 11. 2013/1/23 に PC ゲームをネットで売るサービス (Steam) 運営で有名な Valve 社が、いくつかの Linux 対応のゲームを無料で提供しますと決めました。どんな人が対象でしょうか？
☐ A 全 Debian ユーザ
☐ B 全 Debian 公式開発者
☐ C 全 IT 企業の Debian サーバー戦士

問題 12. Debian の公報チームが、ソーシャルメディアの公式アカウントでの発言する内容の募集をしています。最初に投稿されるのはどのアカウントでしょうか？
☐ A twitter の debian
☐ B google ＋の debian
☐ C identi.ca の debian

問題 13. Debian Member へ SIP サービスが提供されました。Debian Member じゃない人が Debian Member と SIP を使ったコミュニケーションをするときに便利なサイトは？
☐ A rtc.debian.org
☐ B freephonebox.net
☐ C www.nttdocomo.co.jp

問題 14. Debconf14 の開催日は？
☐ A 8 月 23 日〜31 日
☐ B 8 月 10 日〜24 日
☐ C 7 月 21 日〜31 日

問題 15. Jessie のデフォルトの init システムが投票により決定しました。さて何になったでしょう？
☐ A sysvinit
☐ B upstart
☐ C systemd

問題 16. 2014 年 GSoC のメンター募集が行われています。2014 年の GSoC にて採択されていないものはどれ
☐ A hurd-i386 の開発
☐ B clang で Debian のパッケージをコンパイルできるようにする
☐ C Android 上で Debian 環境を作れる件の改良を行う

問題 17. 2014/2/14 にバグレポートの ID が#740000 を向かえました。#730000 からどのぐらいの期間がたったでしょう？
☐ A 1ヶ月と 3 日
☐ B 3ヶ月と 4 日
☐ C 10ヶ月と 10 日

問題 18. Debian のコミュニティにより提供されている Web サービスについて調査が行われています。この調査の名前は？
☐ A Debian Services Servey
☐ B Outreach Program For Women
☐ C Debian Services Census

問題 19. 毎年恒例の DPL 選挙が始まりました。2014 年の DPL 立候補者は誰？
☐ A Takahide Nojima
☐ B Lucas Nussbaum
☐ C Stefano Zacchiroli

問題 20. 2015 年の Debconf15 の開催国はどこになったでしょう？
☐ A ボスニア・ヘルツェゴビナ
☐ B ウクライナ
☐ C ドイツ

問題 21. DPL 選挙が行われました。2014 年の DPL は誰になったでしょうか？
☐ A Lucas Nussbaum
☐ B Neil McGovern
☐ C Raphaël Hertzog

問題 22. clang3.4 による Debian パッケージの再構築が行われました。結果何 % のパッケージが成功したでしょうか？
☐ A 90%
☐ B 50%
☐ C 10%

問題 23. beagle board シリーズという非常に人気のある ARM の実験ボードにバンドルされる OS の将来の見通しについて、beagle board の創立者がどの OS にする予定と発言したか？

☐ A Gentoo

☐ B Debianっしょ！Debian

☐ C Andoroid OS

問題 24. 激論の末、2014/3 中旬頃に Debian の ca-certificates パッケージから消えた root 証明書があります。それはなんでしょう？

☐ A RapidSSL の root 証明書

☐ B CAcert の root 証明書

☐ C Verisign の root 証明書

問題 25. Jessie にてデスクトップ環境を選択した際に導入される、デフォルトコミュニケーションツールの候補について議論がされていります。以下のどれでしょう？

☐ A Empathy

☐ B licq

☐ C jitsi

問題 26. 2014/4/2 に Debian に OTR チームという OTR ソフトをパッケージ化するグループが結成されました。ところで OTRってなんの略？

☐ A Owa-Tte-Ru

☐ B OpticalTRacking

☐ C Off-the-Record

問題 27. 2014/4/16 にて Debian squeeze のサポート期間が伸びる宣言が DSA によりアナウンスされました。結局いつになった？

☐ A 2015/2

☐ B 2016/2

☐ C 2016/5

問題 28. 2014/4/26 に、とあるアーキテクチャが testing から外されました。次のうちのどれでしょう？

☐ A i386

☐ B armel

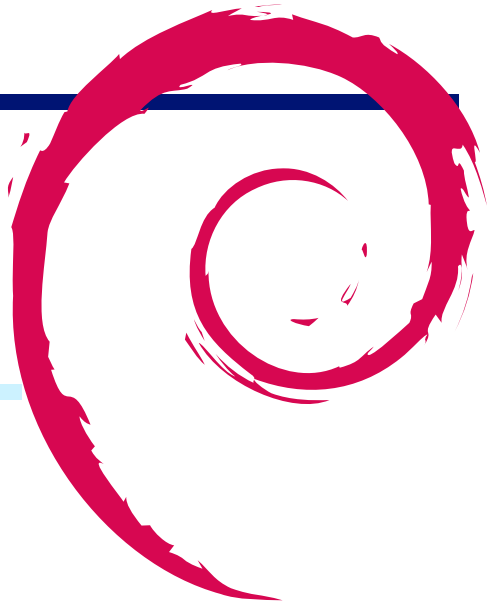
☐ C sparc

問題 29. 2014/4/26 に debian の安定版がリリースされました。バージョンはいくつでしょう？

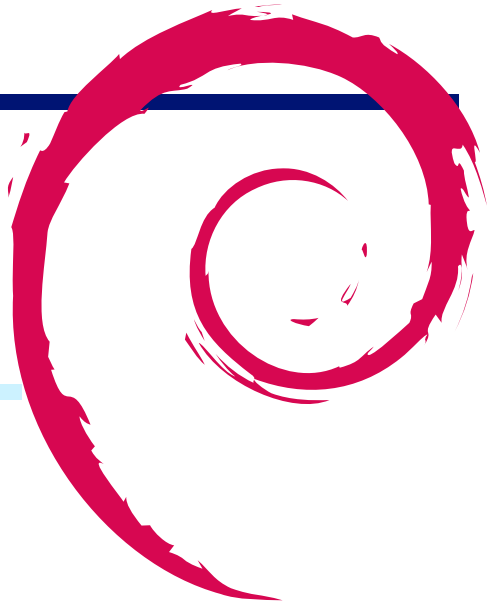
☐ A 7.4

☐ B 7.5

☐ C 7.6



10 索引



11 Debian Trivia Quiz 問題回答

野島 貴英

Debian Trivia Quiz の問題回答です。 あなたは何問わかりましたか？

1. A debian wheezy をお使いの皆様は早速アップデートしましょう！

2. B Outreach Program For Women(略して OPW) は、
`http://gnome.org/opw/` に情報があります。

3. C 元 HP の Linux 部門の CTO を務めたという経歴の持ち主です。debconf とかに参加/ビデオ視聴とかするとわかるのですが、常連さんのようです。

4. C 正しくは”Debian PHP Maintainers” です。以前、Debian PHP Maintainers は、PHP 本体のパッケージも、PEAR モジュールのパッケージも両方メンテナンスしていました。

5. B Debian に貢献した Debian Developer 以外の人のアカウントが `http://contributors.debian.org/` にリストアップされるようになりました。なお、貢献についての集計の元は、`https://contributors.debian.org/sources/` に掲載されている情報を元に集計しているとの事です。

6. A 2013/12/23 現在、powerpc/ia64/sparc アーキテクチャのデフォルト C コンパイラはまだ gcc 4.6 のようです。Debian の次期バージョンの Jessie では gcc 4.6 はサポート対象外なので早いところ gcc 4.6 から脱却する必要があります。

7. A Maria DB は、LAMP システムで有名な Mysql DB の別の実装です。ついに Maria DB キター！今後の Mysql 依存の Debian のパッケージの動向が気になるこの頃です。

8. C 長い間お疲れ！＞ ia64。AMD の戦略に負けた、世間に負けた。ところで、hurd-i386 アーキテクチャと sparc アーキテクチャも、2014/1 時点で Release Team によればわりと崖っぷちな状況のようです。 参考：`https://lists.debian.org/debian-devel-announce/2014/01/msg00008.html`

9. B wheezy 使いの人は早速アップデートだ！今回もセキュリティに関する Bugfix が主です。

10. A 最新版は、`https://wiki.debian.org/Teams/DPL/TrustedOrganizationCriteria` に掲載されています。ちなみに、信頼に足る組織が何をするのかの定義については、Debian プロジェクト憲章の 9.4 章にあります。今まで明確な基準がなかったのか？というのがちょっと驚き。

11. B これもコミュニティへの企業の寄付の方法として面白いと思いました。いわゆる「現物支給」という奴ですな。

12. C debian のソーシャルメディアの公式アカウントから Debian の活動をアピりたい人は応募してみると良いとおもいます。
`https://wiki.debian.org/Teams/Publicity/Identica`

13. B DSA 頑張った!Debian Member の人は `rtc.debian.org` に xxxx @ debian.org を SIP アカウントにしてログインしておくと、Debian Member 以外の人は freephonebox.net から xxxx @ debian.org 宛に連絡を取ることができるとの事。

14. A 2014 年はアメリカ　オレゴン州　ポートランドで開かれます。先日スポンサー募集の案内が流れました。Debconf14 については `http://debconf14.debconf.org/`。Debconf13 の様子は `http://www.irill.org/videos/debconf13` で見れます。

15. C 長い間の論争にケリがついたようです。早速 systemd の使い方を覚えないと。参考:ctte の投票アナウンス `https://lists.debian.org/debian-ctte/2014/02/msg00281.html`、結論 `https://lists.debian.org/debian-ctte/2014/02/msg00405.html`

16. A 他にもいろいろな Project が Debian Project から採択されています。Elektrahttp://www.libelektra.org で設定ファイルのアップグレードを改良するとか、libstdc++ から libc++ を使うように Debian を変更する件や、パッケージ管理に Muon を使う件など。参考：https://wiki.debian.org/SummerOfCode2014/Projects

17. B 毎年、Christian Perrier さんにより、バグレポートの ID について、将来いつ何万番台を迎えるかについて当てるコンテストが行われています。

18. C 2014/2/13 に呼びかけが行われました。現在のサービスの名前と URL のリストは、https://wiki.debian.org/Services にまとめられています。

19. B lucus は 2013 年 DPL ですが、2 年連続立候補となります。他の 2 名の方は、Gergely Nagy さん、Neil McGovern さんとなります。 選挙期間は 2014/3/31〜4/13 となります。各候補者の声明は、http://www.debian.org/vote/2014/platforms/ に掲載予定です。

20. C 2015 年はドイツだそうです。ちなみに 2014 年の Debconf14 は 8/23-31 で USA の Portland,Oregon で開催予定です。

21. A 2014 年の DPL の候補者は 2 名で、Lucas Nussbaum さん、Neil McGovern さんの 2 名でした。 2 名とも自薦とのことです。選挙の結果、Lucas Nussbaum(lucus) さんの圧勝だったようです。ちなみに、Raphaël Hertzog さんは、The Debian Administratorś handbook の作者、他にも偉業がたくさん。

22. A http://clang.debian.net/ が clang による Debian パッケージ再構築のポータルサイトです。毎年、clang のバージョンを上げて、全 Debian パッケージを再構築した結果が載ります。2014/1 の結果としては再構築対象のパッケージの数が 2013/1 より大幅に増えているにもかかわらず、構築失敗に終わったパケージ数が変わらなかったという非常に良い結果となっています。

23. B http://opensource.com/life/14/3/interview-jason-kridner-beagleboard にて、将来 Debian にするという発言があります。ところで、beagle board シリーズは未だに OMAP を使い続けるのかが興味津々ではあります。

24. B 議論のサマリは、LWN の記事 https://lwn.net/Articles/590879/ が判りやすいです。また、CAcertって何？という方は、第 71 回東京エリア Debian 勉強会 (2010 年 12 月開催)http://tokyodebian.alioth.debian.org/2010-12.html に掲載されている勉強会資料がお勧めです。

25. C Debian では、デフォルトのコミュニケーションツールとして、ほぼ完全に RTC/VoIP をサポートすることが望ましいとされたため、こちらに向いているツールとして今までの Empathy よりも jitsi が向いているのでは？ということから議論が開始されました。

26. C Off-the-Record とは、暗号化技術を使ってインフラ提供者にすらメッセージのやりとりの内容を見せない（記録させない）メッセージサービスを目指したものです https://www.otr.im/。

27. B Long Term Support(LTS) だそうです https://lists.debian.org/debian-security-announce/2014/msg00082.html。予定では、2014/5/31 頃にサポート終了するはずでしたので、 2 年弱の延長となります。ただサポート延長されるのは、Debian squeeze の全部のパッケージではないので、サポートされないパッケージを十分にお使いの皆様は、Debian wheezy へアップグレードすることをお勧めしておきます。

28. C リリースチームの見解によれば、ツールチェインの問題、安定性の問題、また今後 Jessie リリースに向けての開発について明確な見解が開発チームらから得られなかったとのことです。

29. B 5 回目の更新リリースとなります。安定版を使っている人でアップデートしていない人は、早速 apt-get upgrade をお勧めしておきます。主にバグフィックスとセキュリテイ対策となります。

『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』（第 107 回から第 113 回）および『関西 Debian 勉強会』（第 83 回）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。第 80 回から第 82 回、第 84 回、関西 Debian 勉強会はもくもく会のため無し、第 110 回東京エリア Debian 勉強会資料はオープンソースカンファレンス 2014 Tokyo/Spring のため無し、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 2014 年夏号

2014 年 8 月 12 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）