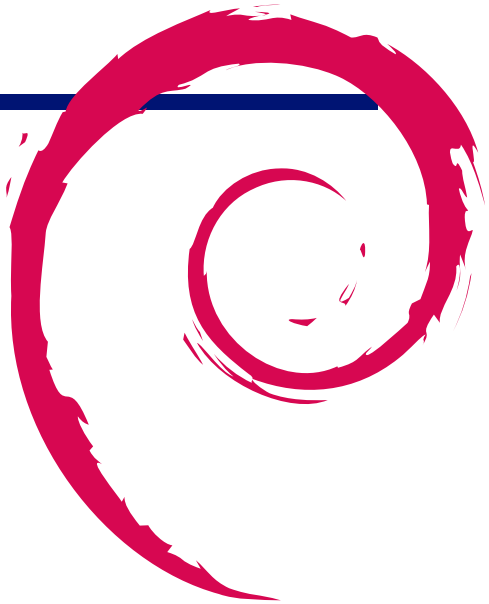
.Deb

銀河系唯一のDebian専門誌

2012 年 11 月 17 日

特集: Linux perf

特集: Bluetooth Tether



1 はじめに

上川 純一

今月の 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?

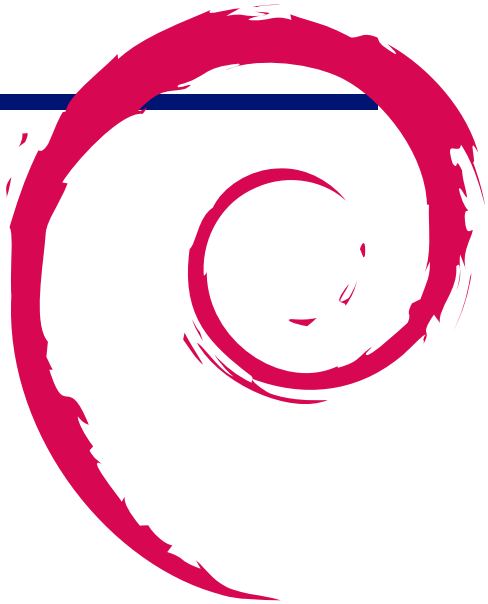
Debian 勉強会の目的は下記です。

- 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。

デビアン勉強会

目次



2 事前課題

上川 純一

今回の事前課題は以下です:

1. Debian で今いまいちサポートされていない機能、いけてない実装について語ってください。

この課題に対して提出いただいた内容は以下です。

2.1 koedoyoshida

- Wheezy インストーラのパーティション構成時のデータ消去待ち時間。暗号化パーティションを作ろうとすると果てしなく待たされる。
- Wheezy の壁紙。いけてない感じで OSC とかの展示で適当なものを選ぶのが面倒で結局 squeeze や Debian の (過去の) ジェネリックなものを選ぶことに...
- デバッグシンボルを含んだバイナリがない。以前大統一で岩松さんが発表していた話が進んでいるとうれしい。

2.2 キタハラ

私が使用する範囲ではありません。

2.3 MATOHARA

あまり思いつかないですが、人と話をしているとき以下のようなことを言われたことがあります。

- Debian は規定値の設定がいけてないので設定を沢山いじらないと運用出来なくて工数が無駄に掛かる
- Debian はインストールが難しい

具体例を聞けなかったのですが、設定については千差万別なのでその人にとって向いていなかったからと言ってダメかというとそうではないと思います。しかし、 openSUSE の YaST は一元的に管理できて便利そうだなとは思います。インストールが難しいというのも昔のイメージなのか現在のことを言っているの不明なのですが、デスクトップ向けのディストリビューションに比べると選択肢が多いので難しく感じられるのかもしれません。

2.4 鈴木崇文

現在は改善しているかもしれませんが、一部 kernel のパッケージによっては dbg パッケージが無いものがあったりして、困った経験がありました。

2.5 野島 貴英

Debian でいけていない機能／実装といわれると、

- ifupdown パッケージ
- Solaris10 以上でいうところの FMD とか、 SVC 欲しい。
- インストーラで'/' 全部 BTFS というの選択可能でしたっけ?
- WEB 絡みで、最新 WEB 開発関係一式のパッケージリポジトリというのが欲しい気がする(WEB システムで流行りものやら、良く使われていそうなバージョンのソフトに特化したリポジトリ。それなら experimental しか存在しないとかでもイイ!)
- Debian とはちょっとズレてるかもしれませんが、synaptic は...iTunes みたいになってほしー

と言いたい放題言ってみた。

2.6 上川純一

Android 用の ADB コマンドとかが標準で入っていると嬉しいなぁ。

2.7 yamamoto

Debian の理想と信念は大好きなんですが、少し気になる点もあります。

例えば Debian-Installer には non-free で頒布されている firmware パッケージが含まれていない点などです。 non-free のパッケージは、様々な理由で non-free に分類されているわけですが、その一つにテキストのソースが存在しないため、とかいう理由もあります。

別に「non-free にするな」とか主張したいわけではないのですが、頒布の制限の無いパッケージまで「non-free だから」と、収録を拒絶するのは少々やりすぎではないかと考えています。

2.8 野首

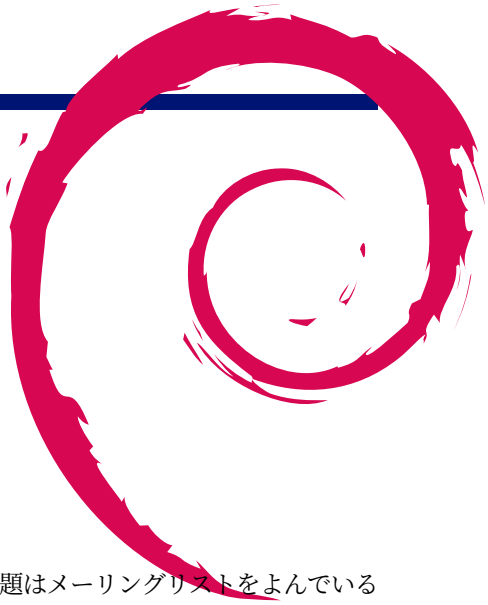
- パッケージのリストアに dpkg –get-selections はちょっと微妙
- aptitude-run-state-bundle はいまいち用途がわからない
- stable にたまに使い物にならないパッケージがある
  - 古すぎるからとか (例: lxc)
- パッケージのトランザクションが欲しい
  - 一度インストールしてみておかしかったら revert
  - 問題なければ commit みたいな
- multiarch は導入して本当に良かったのか?

2.9 日比野 啓

使いこみが足りてないのかもしれませんが、履歴管理システムに保存されている tree を debian パッケージとして build したり install したりするときに便利なツールがあまり無いのかもと思いました。

2.10 dictoss(杉本　典充)

iptables のコマンド引数が他の OS と違うような感じがする。そのため iptables の初心者が web で調べたコマンドを実行しても構文エラーではじかれて辛い。



3 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. FTP master にあたらしく参加したのは

☐ A iwamatsu

☐ B ansgar

☐ C bdale

問題 2. pdiff で何が改善されたか

☐ A 最大 2 つの Diff をダウンロードすれば良いように変更になった

☐ B 一日 10 個づつ Diff を生成するようになった

☐ C Diff ってなにそれおいしいの?

問題 3. CTTE 573745 で何が決定されたか

☐ A Mattias Klose クビ

☐ B python 終了のお知らせ

☐ C みんな仲良くしようね

問題 4. 新しく Front Desk のメンバーになったのは

☐ A Kouhei Maeda

☐ B Iwamatsu

☐ C Jonathan Wiltshire

問題 5. debian installer 7.0 beta3 の新機能ではないのはどれか

☐ A ipv6

☐ B UEFI

☐ C grub2

問題 6. Debconf13 はどこで開催されるか

☐ A スイス

☐ B 日本

☐ C 中国

問題 7. debian-cloud は何をするメーリングリストか

☐ A 人をけむにまくため

☐ B クラウドサービスで Debian を利用する

☐ C エアリスー

問題 8. Official Logo が変更されたのはなぜか

☐ A 古い Official Logo が DFSG Free じゃなかったから

☐ B 時代に合わなくなってきたから

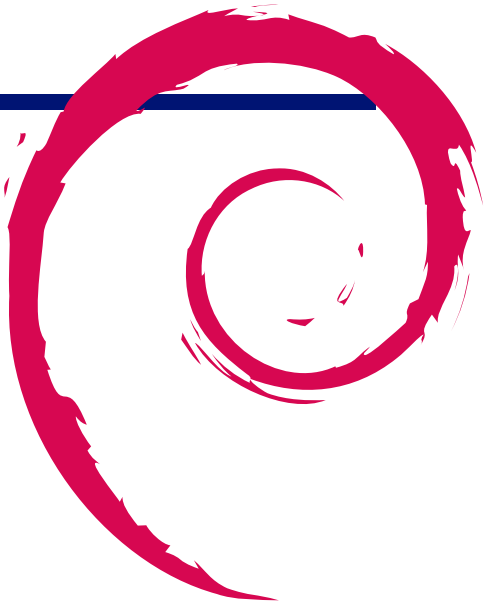
☐ C DPL の趣味

問題 9. codesearch.debian.net は何をするサービスか

☐ A 正規表現でソースを検索できる

☐ B ソースコードクレクレ

☐ C ブログサービス



4 Android 携帯で Bluetooth Tethering

上川純一

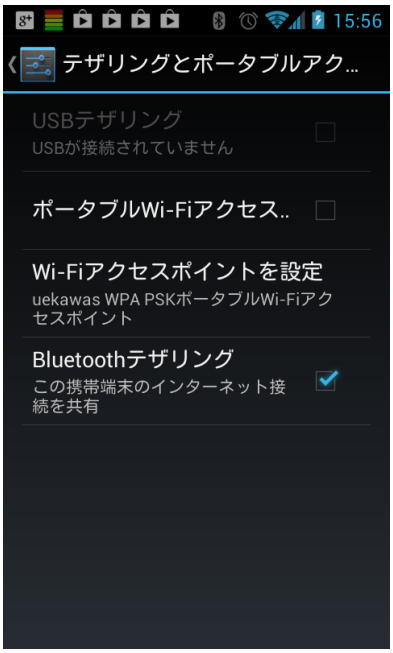
4.1 はじめに

最近の Android 携帯もパソコンも Bluetooth に対応しています。そして Bluetooth profile である PAN や DUN に対応しているものも多いようです。 Android 4.0 あたりで対応するようになったと思われる Bluetooth Tethering を利用することで Android 携帯に Bluetooth PAN 経由で接続し、 Android 携帯の回線を利用して外部のネットワークに接続できるようになります。

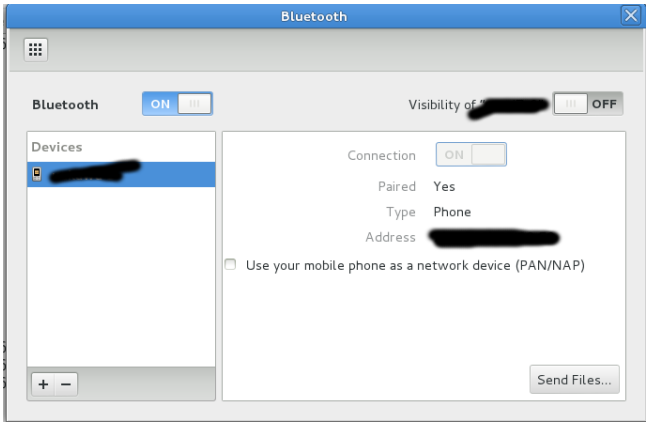
WiFi Tethering のように電力消費が高すぎるので毎回設定でオフにする必要があるるということもなく、や USB Tethering のように毎回物理的に接続する必要もないので便利です。カバンの中に携帯をいれたまま接続できるので気楽ですよ。

4.2 設定方法

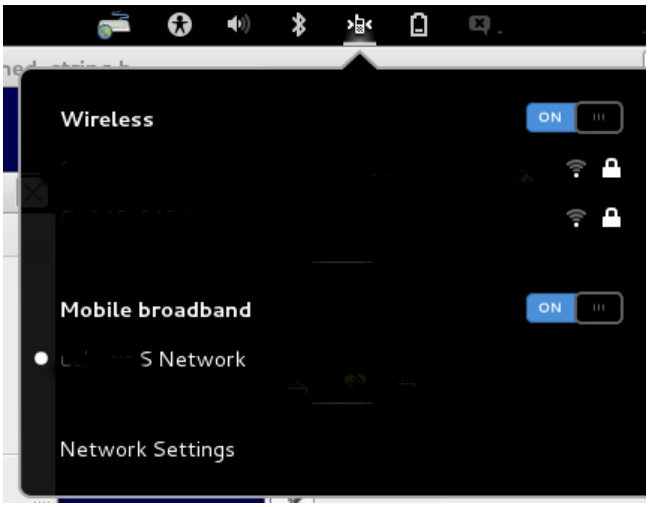
Android 携帯側では Bluetooth Tethering をオンにします。図 4.2)



あと、 Bluetooth のペアリングを行います。携帯電話側の設定で可視状態にしておいて GNOME の設定で追加すればよいでしょう。 (図 4.2)



一旦設定しておくとネットワークの選択候補に Mobile Broadband というのが現れてそこで携帯電話の Bluetooth PAN 接続が選択できるようになります。



4.3 ネットワーク構成

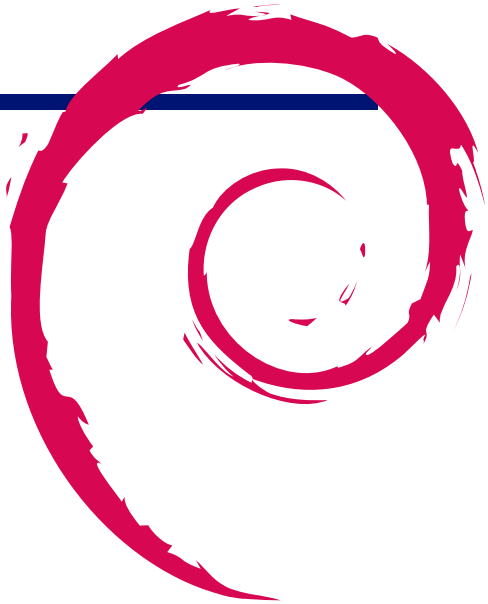
Linux 側からは bnep0 デバイスとして見えます。複数マシンから接続するとそれぞれが別のサブネットに接続されるっぽいのでお互いに通信はできないようです。 WiFi Tethering だと同じサブネットにつながるので個人的にはウェブサーバーとクライアントを接続するためのハブとして便利に利用していたのですが、そういう使い方はできないようです。

```
bnep0     Link encap:Ethernet  HWaddr
          inet addr:192.168.46.43  Bcast:192.168.46.255  Mask:255.255.255.0
```

4.4 まとめ

Bluetooth PAN は便利、ということでした。ただ、なぜだか日本の携帯電話は Bluetooth PAN に対応してないものが多く、また海外携帯の日本モデルはその機能を削ってたりするものがあります[*1]。ここはぜひ Bluetooth PAN 対応じゃない携帯電話は購入しないことで消費者の意見を表明してください。

[*1] 例: 筆者の所有する Galaxy Nexus DoCoMo 版



5 perf でパフォーマンスチューニング

上川純一

5.1 はじめに

最近のたいていの CPU には Hardware Performance Counters という仕組みがあり、特定のイベントが一定回数発生したらある処理をするという事ができるようになっています。それを利用してプロファイラーが実装できて、プログラムのパフォーマンスボトルネックの発見に役立てることができます。昔は oprofile を使っていたのですが、最近の Linux カーネルでは perf という仕組みを使うのが主流のようです。

カーネル側のサポートは標準で入っています。コマンドラインの perf コマンドは linux-base パッケージにはいっていてたいていの環境では標準でインストールされているようにみえるのですが実体はカーネルにあったバージョンを追加でインストールする必要があります。例えば、 linux 3.2 だと linux-tools-3.2 をインストールすることになります。

```
 $ uname -r
3.2.0-3-amd64
 $ sudo apt-get install linux-tools-3.2
```

デバッグシンボルがあると関数名とかがきれいに出る気がするので利用しているライブラリのデバッグシンボルもいれておくとよいでしょう。標準ライブラリはいれておきましょう。

```
 $ sudo apt-get install libc6-dbg libstdc++6-4.7-dbg
```

5.2 perf stat

プログラム単体の実行時間を計測してレポートしてくれるコマンドとして time コマンドがありますが、その代わりにつかえそうなツールとして、 perf stat があります。最近の CPU は負荷によって CPU 周波数が変わり、そういうシステムにおいてパフォーマンスの測定のために実行時間だけを計測するというのは適切ではないのですが、 perf stat はそれ以外に必要そうな値を計測してくれるので便利です。

```
$ perf stat ./apt-index-cmd debian_dists_sid_main_binary-amd64_Packages debian > /dev/null
 Performance counter stats for './apt-index-cmd debian_dists_sid_main_binary-amd64_Packages debian':

       1741.828818 task-clock                #    0.997 CPUs utilized
               165 context-switches          #    0.000 M/sec
                 6 CPU-migrations            #    0.000 M/sec
            27,392 page-faults               #    0.016 M/sec
     4,990,934,326 cycles                    #    2.865 GHz                     [83.29%]
     1,681,297,382 stalled-cycles-frontend   #   33.69% frontend cycles idle    [83.27%]
     1,096,373,883 stalled-cycles-backend    #   21.97% backend  cycles idle    [66.62%]
     7,738,965,303 instructions              #    1.55  insns per cycle
                                             #    0.22  stalled cycles per insn [83.51%]
     1,784,494,907 branches                  # 1024.495 M/sec                   [83.49%]
        32,701,183 branch-misses             #    1.83% of all branches        [83.32%]

       1.746581711 seconds time elapsed
```

5.3 perf record

perf record コマンドはプロファイル情報を記録する命令です。パラメータとして指定したコマンドをそのまま実行して、カレントディレクトリに perf.data ファイルを作成します。後に perf report などでそのプロファイルデータを確認することができます。

```
$ perf record ./apt-index-cmd debian_dists_sid_main_binary-amd64_Packages  debian > /dev/null
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.087 MB perf.data (~3800 samples) ]
$ ls -l perf.data
-rw------- 1 dancer dancer 93560 10 月 10 07:03 perf.data
```

perf record -g オプションをつけるとコールグラフ情報も記録するようです。

デフォルトは一秒 1000 サンプルとるようなので、プログラムの実行時間に応じてサンプル数を適当に調整しましょう。例えば全体で実行が一秒で終わってしまうプログラムの場合は-F 10000 をつけてみるとよりサンプル数がとれていいかもしれません。

5.3.1 gcc のコンパイルオプションの検討

gcc でソースコードコンパイルするときに、-g オプションをつけるとデバッグ情報がつきます。関数シンボルの情報とか行数とかソースコードとかが得られるのでこれは多分重要。

コールグラフがあまりないなぁと思ったら最適化のしすぎとスタックトレースのとりにくさを疑ってみましょう。

通常最適化オプション -O2 をつけてコンパイルしますが、 amd64 の場合、 gcc で-O2 をつけてコンパイルするとフレームポインタがなくなってスタックトレースを取得しにくくなっています。 libunwind 使えば取得できるはずですが、多分 perf はカーネル空間でスタックトレースをとっているのでそうなってないっぽいです。対策として若干オーバーヘッドがありますが、-fno-omit-frame-pointer を指定すると、フレームポインタを操作するコードを生成してくれます

C++ でファンクションオブジェクトとかテンプレートプログラミングしまくっていると、関数がほとんどインライン展開されてしまい、コールグラフに現れる部分が少なくなります。理解不能になってきたらたまに -O あたりでコンパイルしてスタックトレースを見てみましょう。

5.4 perf report

perf report コマンドは取得したプロファイルデータを可視化するコマンドです。テキストコンソールメニュー形式になっていて、気になるシンボルを選択してソースコードのアノテーションを見ることができます。どのソースコードの行に対応するアセンブラのどの命令で CPU 処理時間を費やしたのかを表示してくれます。

```
$ perf report
Events: 1K cycles
 18.65%  apt-index-cmd  apt-index-cmd        [.] available_parser::AptIndexSpiri
  7.38%  apt-index-cmd  libc-2.13.so         [.] _int_malloc
  6.64%  apt-index-cmd  libc-2.13.so         [.] malloc
  5.17%  apt-index-cmd  libstdc++.so.6.0.17  [.] std::string::_M_replace_aux(uns
  4.03%  apt-index-cmd  libc-2.13.so         [.] _int_free
  4.02%  apt-index-cmd  libstdc++.so.6.0.17  [.] __cxxabiv1::__vmi_class_type_in
  3.81%  apt-index-cmd  libstdc++.so.6.0.17  [.] std::string::_M_mutate(unsigned
  3.59%  apt-index-cmd  libstdc++.so.6.0.17  [.] std::ctype<char> const& std::us
  3.00%  apt-index-cmd  libc-2.13.so         [.] __strcmp_sse42
  2.74%  apt-index-cmd  apt-index-cmd        [.] boost::detail::function::functi
  2.72%  apt-index-cmd  libstdc++.so.6.0.17  [.] __dynamic_cast
  2.67%  apt-index-cmd  libc-2.13.so         [.] free
  2.46%  apt-index-cmd  libc-2.13.so         [.] __memcmp_sse4_1
```

ここでエンターをおすと次が表示され

```
available_parser::AptIndexSpirited::MakeIndex()::{lambda(std::vector<char, std::
    0.00 :          40368a:       je     4036ac <available_parser::AptIndexSpir
         :
         :          const std::string& get() const { return my_string_; }
         :
         :          // for 'map' comparison.
         :          bool operator<(const OrderedHashedString& b) const {
         :            if (ordered_hash_ == b.ordered_hash_) {
    2.87 :          40368c:       mov    0x28(%rbx),%rdx
         :              return my_string_ < b.my_string_;
         :            } else {
         :              return ordered_hash_ < b.ordered_hash_;
   34.96 :          403690:       cmp    %rbp,%rdx
    1.43 :          403693:       setb   %cl
         :
         :          const std::string& get() const { return my_string_; }
         :
         :          // for 'map' comparison.
         :          bool operator<(const OrderedHashedString& b) const {
         :            if (ordered_hash_ == b.ordered_hash_) {
```

5.4.1 C++ のコードの場合

C++ でテンプレートを活用しているコードを眺める場合、 perf report の TUI インタフェースだと関数名が十分表示されないなと悩むことになります。とりあえず TUI じゃないインタフェースにするともっと表示されますが、いまいち全体は表示できません。

たとえば関数名がこのように途中で切れてしまいます。まだ僕は回避方法を発見していません。

```
void
MeasureRaw<std::unordered_map<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> >, int,
RangeHash<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> > >,
RangeEqualTo<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> > >,
std::allocator<std::pair<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> > const, int> > >, __gnu_cxx::__normal_iterator
```

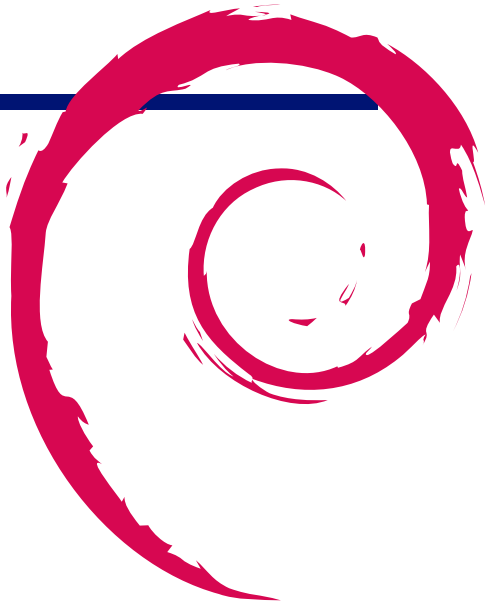
5.4.2 おわりに

一番基本的な Perf の使い方を紹介しました。 sudo perf list の出力を確認するとハードウェアのいろいろなカウンタだけでなくカーネルが提供しているさまざまなトレースポイントでのカウンタがあるのがわかります。どう使うか、夢は広がりますね。

参考文献

[1] perf(1) "perf – perfomance analysis tools for Linux" manual page. perf_3.2-report(1), etc.

[2] "Tutorial – Linux kernel profiling with perf", `https://perf.wiki.kernel.org/index.php/Tutorial`



6 systemd

岩松 信洋

6.1 はじめに

世の中の主要な Linux ディストリビューションは SysVinit の init scripts から他の init システムに移行しつつあります。 Fedora や Arch Linux が systemd に移行を始めたということもあり、一部で盛り上がっている（ 阿鼻叫喚ともいう）ようです。まさか いまだに SysVinit を使っている Debian 勉強会参加者がいるとは思えませんが、盛り上がっているようなので Debian と Systemd についてまとめてみました。

6.2 systemd とは?

RedHat に勤めている Lennart Poettering 氏によって開発されている init の代替プログラムです。実際には init の代替だけではなく、 Linux のサービス（ デーモン）管理フレームワークとなっています。今までの init システムの違いはサービスのプロセス管理を pid ではなく、 cgroups を使う点とサービスの起動をソケットとバスを使う点があります。これらは D-Bus を使って行います。これによってシステム立ち上げ処理をより並列的に行えるようになっています。また、System V スタイルと BSD スタイルの両方をサポートしています。

開発は freedesktop.org `http://cgit.freedesktop.org/systemd/systemd/` で行われており、開発は活発で週に一度はバージョンアップしています。最新バージョンは v195 となっています。

6.2.1 systemd の利点

systemd は SysVinit と比べて次のような利点があります。

- 設定が容易。
  SysVinit はシェルスクリプトで記述していたため、開発者によって書き方が異なります。よって設定や内容の理解が難しいことがあります。 systemd は設定方法や項目等が決まっているため、設定しやすくなっています。
- 起動が早い。
  シェルに依存していないのとデーモンが並列起動するため起動が早いです。
- カーネルモジュールの操作、セッション管理、ログ管理、ディスクの暗号化などを統合。

その他、作者による説明を `http://0pointer.de/blog/projects/why.html` から参照できます。

6.3 Debian で使う

systemd はもちろん Debian でも提供されており、 testing / unstable で v44 が利用できます。最新版とバージョンに差がありますが、アップストリームで頻繁にバージョンアップするのでバージョンはあまり問題ではありません。 v44 でも systemd を十分に使うことができます。

いまのところ Debian に関する情報は http://wiki.debian.org/systemd にまとまっていますが情報が少なく、内容も古いです。

6.3.1 インストール

先にも書いたように Debian では v44 が最新版です。 apt-get / aptitude でインストールできます。

また、 Linux カーネルは 2.6.39 以上、 devtmpfs, fanotify, autofs4, cgroups が有効になっている必要があります。

インストールは以下のように実行します。

```
$ sudo apt-get install systemd
```

以下のパッケージが依存関係でインストールされます。

```
libsystemd-daemon0
libsystemd-id128-0
libsystemd-journal0
libpam-systemd
```

次に ブートローダに init 指定を追加します。 grub を使っている場合、/etc/default/grub の GRUB_CMDLINE_LINUX_DEFAULT に **init=/lib/systemd/systemd** を追記します。

```
変更前:
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
変更後:
GRUB_CMDLINE_LINUX_DEFAULT="quiet init=/lib/systemd/systemd"
```

変更後、 **update-grub** を実行し、 grub に設定を反映します。そしてリブートします。設定が間違っていないければ systemd で立ち上がるはずです。

```
$ sudo update-grub
.....
$ sudo reboot
```

6.3.2 起動速度

systemd はアナライザをデフォルトでサポートしています。起動にかかった時間を確認するには systemd-analyze を実行します。また、画像で確認したい場合には prop オプションを指定して実行します。 SVG フォーマットで出力されるので、リダイレクトしてファイルに保存します。

```
$ systemd-analyze
Startup finished in 1831ms (kernel) + 5669ms (userspace) = 7500ms
$ systemd-analyze plot > systemd-boot.svg
```

試しに自分が常用している環境で起動時間を測定したところ、 SysVinit は約 15 秒、 systemd は約 10 秒でした。

6.4 用語

systemd を扱っていると専門用語が出てきますので、説明します。

- ユニット
  systemd ではデーモンなどの制御対象のことをユニットと呼びます。ユニットにはサービス、デバイス、マウントポイントなど、いくつかの種類があります。このユニットはテキストファイルで記述され、/lib/systemd/system/ 以下に格納されています。各ユニットは拡張子を持ち、サービスの場合は.service となっています。
  mount, swap, automout は起動時に /etc/fstab から自動的にユニットを生成してくれます。
- ターゲット
  ターゲットとは SysVinit の runlevel 相当のものです。これはディストリビューションによって異なります。Debian の場合は以下のようになっています。
  この他に graphical.target と emergency.target があります。前者は X による起動を行うときに呼ばれるター

ユニットの種類	説明
service	デーモン
socket	ソケットによるデーモン
target	multi-user.target
device	udev で管理するデバイス
snapshot	ある時点の init の状態
timer	イベントから時間経過
path	監視するパス
mount	マウントポイント
swap	スワップ
automaount	自動マウントポイント

表 1　systemd で提供するユニット

run level	systemd のターゲット
0	poweroff.target
1	rescue.target
2 - 5	multi-user.target
6	reboot.target

表 2　run level とターゲットの対応

ゲット、後者は障害が起こった時に起動できるようにするためのターゲットです。ターゲットはカーネルのブートオプションに `systemd.unit=`で指定できます。何も指定しない場合は default.target が呼ばれるようになっています。

6.5　ユニットの操作方法

systemd に移行した後、デーモン等の制御は /etc/init.d/ 以下を実行するのではなく、**systemctl** コマンドを使って操作します。以下にユニットの操作方法について説明します。

6.5.1　起動しているユニットを表示する

起動しているユニットを表示するには sytemctl を実行します。

```
$ systemct
...
console-setup.service     loaded active exited        LSB: Set console font and
cron.service              loaded active running       LSB: Regular background pr
dbus.service              loaded active running       D-Bus System Message Bus
debian-fixup.service      loaded active exited        Various fixups to make sys
exim4.service             loaded active running       LSB: exim Mail Transport A
getty@tty1.service        loaded active running       Getty on tty1
ifup@eth0.service         loaded active exited        ifup for eth0
...
```

6.5.2　全てのユニットを表示する

操作できるユニットを表示するには `--all` を指定します。

```
$ systemctl --all
UNIT                       LOAD   ACTIVE   SUB       JOB DESCRIPTION
proc-sys...misc.automount loaded active   waiting       Arbitrary Executable Fil
dev-cdrom.device          loaded active   plugged       QEMU_DVD-ROM
dev-disk...QM00003.device loaded active   plugged       QEMU_DVD-ROM
dev-disk...QM00001.device loaded active   plugged       QEMU_HARDDISK
dev-disk...2dpart1.device loaded active   plugged       QEMU_HARDDISK
dev-disk...2dpart2.device loaded active   plugged       QEMU_HARDDISK
...
```

6.5.3 ユニットの状態を確認する

ユニットの状態を確認するには、 status オプションに確認したいユニット名を指定して実行します。

以下に rsyslog.service ユニットの状態を確認する例を示します。

```
$ systemctl status rsyslog.service
    Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled)
    Active: active (running) since Wed, 14 Nov 2012 00:37:18 -0800; 22h ago
   Process: 474 ExecStartPre=/bin/systemctl stop systemd-kmsg-syslogd.service (code=exited, status=0/SUCCESS)
  Main PID: 483 (rsyslogd)
    CGroup: name=systemd:/system/rsyslog.service
            └ 483 /usr/sbin/rsyslogd -n -c5
```

これにより、このユニットは /lib/systemd/system/rsyslog.service によって Wed, 14 Nov 2012 00:37:18 -0800 に起動していることが分かります。

6.5.4 ユニットを起動する

起動していないユニットを起動するには、 start オプションにユニット名を指定して実行します。これは /etc/init.d/**サービス** start と同様の動きとなります。

```
$ sudo systemctl start ユニット名
```

6.5.5 ユニットを停止する

起動しているユニットを停止するには、 stop オプションにユニット名を指定して実行します。これは /etc/init.d/**サービス** stop と同様の動きとなります。

```
$ sudo systemctl stop ユニット名
```

6.5.6 ユニットの設定を再読み込みする

ユニットの設定を再読み込みするには、 daemon-reload オプションにユニット名を指定して実行します。

```
$ sudo systemctl daemon-reload ユニット名
```

実際に動いているデーモンの設定、例えば httpd の設定を再読み込みし、再起動するには reload オプションを使います。

6.5.7 ユニットの自動起動を有効にする

ユニットの自動起動を有効にするには enable オプションにユニット名を指定して実行します。

有効にすると /etc/systemd/system/**ターゲット**.wants/に/lib/systemd/system/にあるユニットへのシンボリックリンクが作成されます。どのターゲットで自動起動が有効になるかは、ユニットファイルの Install セクションで指定します。

```
$ sudo systemctl enable ユニット名
```

6.5.8 ユニットの自動起動を無効にする

ユニットの自動起動を無効にするには disable オプションにユニット名を指定して実行します。無効にすると、/etc/systemd/system/ターゲット.wants/にあるシンボリックリンクが削除されます。

```
$ sudo systemctl disabe ユニット名
```

6.5.9 ユニットの詳細を確認する

ユニットの詳細を確認するには show オプションにユニット名を指定して実行します。これにより指定したユニットと他のユニット、ターゲットの関係などが分かります。

```
$ sudo systemctl show rsyslog.service
Id=rsyslog.service
Names=syslog.service rsyslog.service
Requires=basic.target
Wants=syslog.socket
WantedBy=multi-user.target
Conflicts=shutdown.target
...
```

6.6 ユニットについて

ユニットには各ユニット間の依存関係を記述することができます。依存関係の指定として以下があります。

定義	説明
Before	そのユニットの後に起動されるべきユニット。
After	そのユニットの前に起動されるべきユニット。
Conflicts	同時に起動できないユニット。
Service	ソケットによる起動を行うユニット。
Sockets	ソケットによるユニットの起動を行う場合のソケット情報
Wants	同時に起動してほしいユニット。成功、失敗は関係ない。
Requires	同時に起動されなければならないユニット。ユニットの起動が失敗した場合は要求元も失敗する
BindTo	ユニットをグループとしてまとめる。

表 3　ユニットの依存定義

例えば、 default.target の内容は以下のようになっています。

```
[Unit]
Description=Graphical Interface
Requires=multi-user.target
After=multi-user.target
Conflicts=rescue.target
AllowIsolate=yes
```

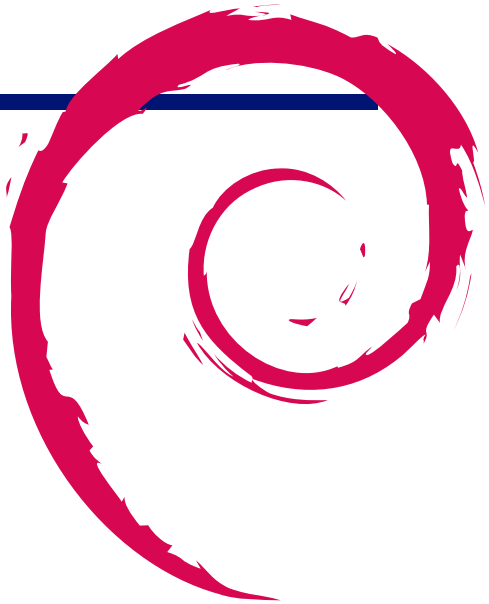
このターゲットは multi-user.target と同時に起動され、 multi-user.target の後に起動します。また、 rescue.target と同時に起動できません。

6.7 まとめ

Debian でも問題なく systemd が利用できる環境が整っています。レガシーな SysVinit は捨て、新しい init の世界へ足を踏み入れてみてはいかがでしょうか。

6.8 参考文献

http://www.slideshare.net/moriwaka/systemd



7 索引

Debian 勉強会資料

2012 年 11 月 17 日　　初版第 1 刷発行

東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）