東京エリア Debian 勉強会

第 135 回 2016 年 1 月度

野島貴英

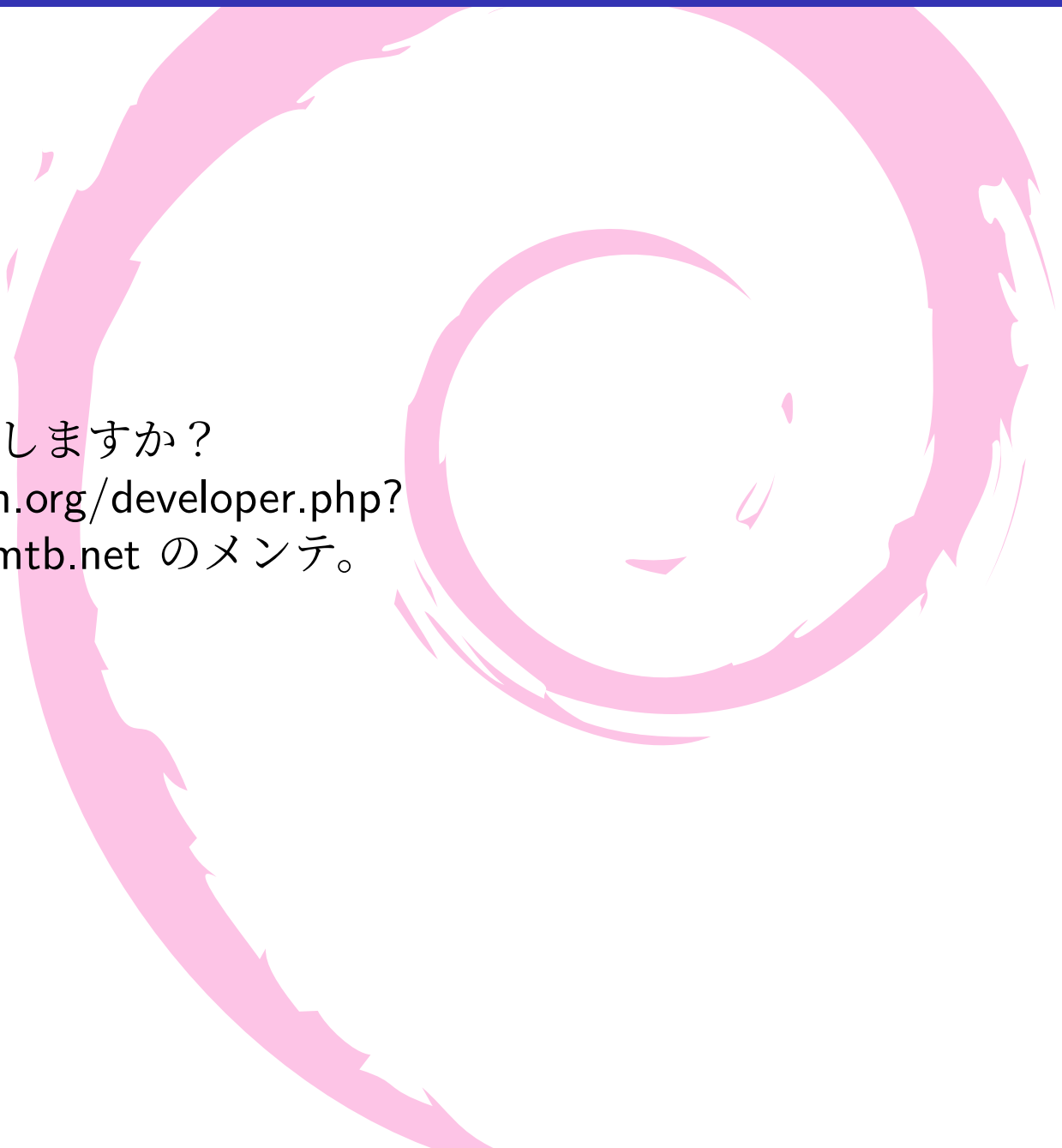
2016 年 1 月 23 日

Agenda

- 事前課題発表
- 最近あった Debian 関連のイベント報告
  - 第 134 回東京エリア Debian 勉強会
- Debian Trivia Quiz
- 2016/6 末までの皆さんの Debian プロジェクト
- Debian で、Linux ftrace まわりをいじってみた
- 今日の宴会場所

事前課題

mkouhei

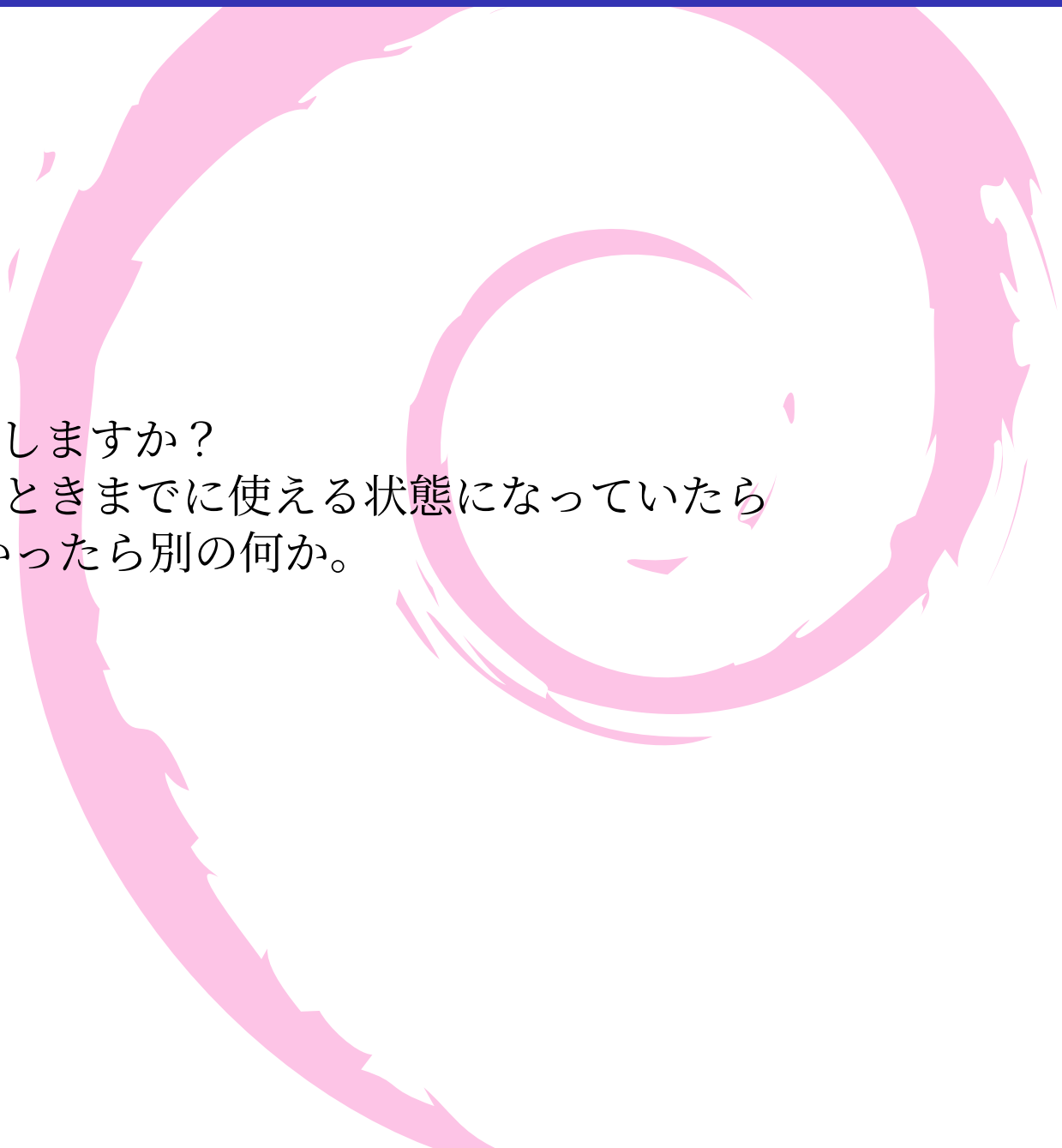


1. Q.hack time に何をしますか？
   A. https://qa.debian.org/developer.php?login=mkouhei@palmtb.net のメンテ。

henrich

1. Q.hack time に何をしますか？
   A. afdko 関連のパッケージか、os-autoinst のパッケージあたりを進めておきたい

kenhys



1. Q.hack time に何をしますか？
   A. porterbox がこのときまでに使える状態になっていたら #770243、じゃなかったら別の何か。

wskoka

1. Q.hack time に何をしますか？
   A. tilegx の debian 化

rosh

1. Q.hack time に何をしますか？
   A. smali のパッケージングを再挑戦します。

dictoss

1. Q.hack time に何をしますか？
   A. bluetooth テサリングの設定を調べて、raspberry pi をネットワークへ接続できるようにする

y.y

1. Q.hack time に何をしますか？
   A. Caff

yy_y_ja_jp

1. Q.hack time に何をしますか？
   A. DDTSS

野島

1. Q.hack time に何をしますか？
   A. DDTSS、東京エリア debian 勉強会のレジメ作り。

イベント報告

第 134 回東京エリア Debian 勉強会

- 場所は dots さんをお借りしての開催でした。
- 参加者は 9 名でした。
- セミナ内容は、2 本建てで、
  1. 参加者皆さんによる「 Debian おすすめのパッケージを参加者に聞いてみた 」
  2. 野島さんによる「 Debian モバイル wifi ルータ化 」

  でした。
- 残りの時間で hack time を行い、成果発表をしました。

第 134 回東京エリア Debian 勉強会 (つづき)

「Debian おすすめのパッケージを参加者に聞いてみた」は、参加登録する際のアンケートにて、おすすめのパッケージとおすすめポイントを記入してもらったものを、皆さんに発表頂きました。10 個のパッケージと 1 つのお勧めソフトウェアが紹介されました。

第 134 回東京エリア Debian 勉強会 (つづき)

「Debian モバイル wifi ルータ化」は、Debian sid をモバイル wifi ルータにしてしまう方法についての発表でした。pppd,bridge,dnsmasq による dhcp サーバ、hostapd を一気に紹介し、はまりどころについて語られました。また、wifi のハードウェアに linux からアクセスする時に利用される、今時のフレームワークについても説明が行われました。

第133回東京エリアDebian勉強会(つづき)

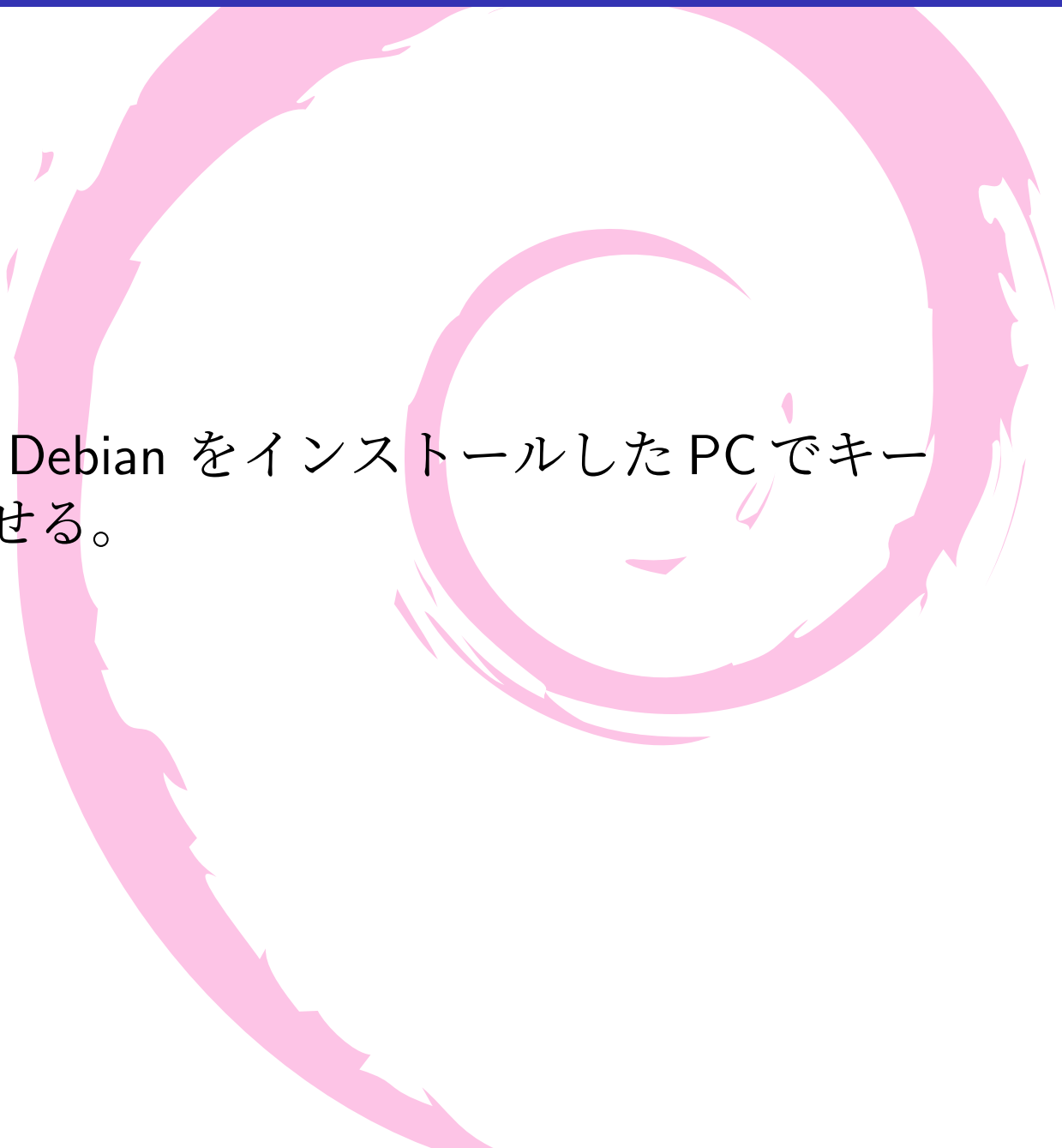
Debianは、デスクトップに、組み込み機材に、様々に自由に使うことができます。様々なパッケージを組み合わせると、民生品と同等の機能を持つものを簡単に作ることができてしまいます。もし、民生品で気に入らないところがあれば、自分で直すこともできるようになります。どこまでも自由なDebianを使って、自分の思い通りの機材を作る方が益々増えるとよいですね。

2016/6末までの皆さんのDebianプロジェクト

はじめに

2016 年も始まりました。参加登録時の事前アンケート機能を用いて、2016/6 末までに、皆さんは Debian について何をするのか？について予定を立てていただきました。今回はこちらを皆さんに発表いただきます。

mkouhei



長女 (3 月で 4 歳）に Debian をインストールした PC でキーボード操作を習得させる。

henrich

ウェブサイトの再構築提案したい。

kenhys

Debian 勉強会にできるだけ参加したい。

wskoka

ports にエントリーしたい

rosh

- Linkstation の kernel DTS をメンテする
- Linkstation の d-i (installer) を対応する

dictoss

kfreebsd を使ってバグ出しする。

y.y

たまっている、未処理 key signing を処理する。

yy_y_ja_jp

- パッケージ更新
- ITP

野島

- 毎月１回は東京エリア Debian 勉強会を開催する
- 手元の Nook HD+にてカスタム Kernel をネイティブで動かし、Debian の root ファイルシステムを認識させる。

おわりに

　様々な目標を立てていただきました。2016/7 にて進捗を確認してみようと思いますので、目標達成に向けて、みなさん頑張りましょう。

Debian で、Linux ftrace まわりをいじってみた

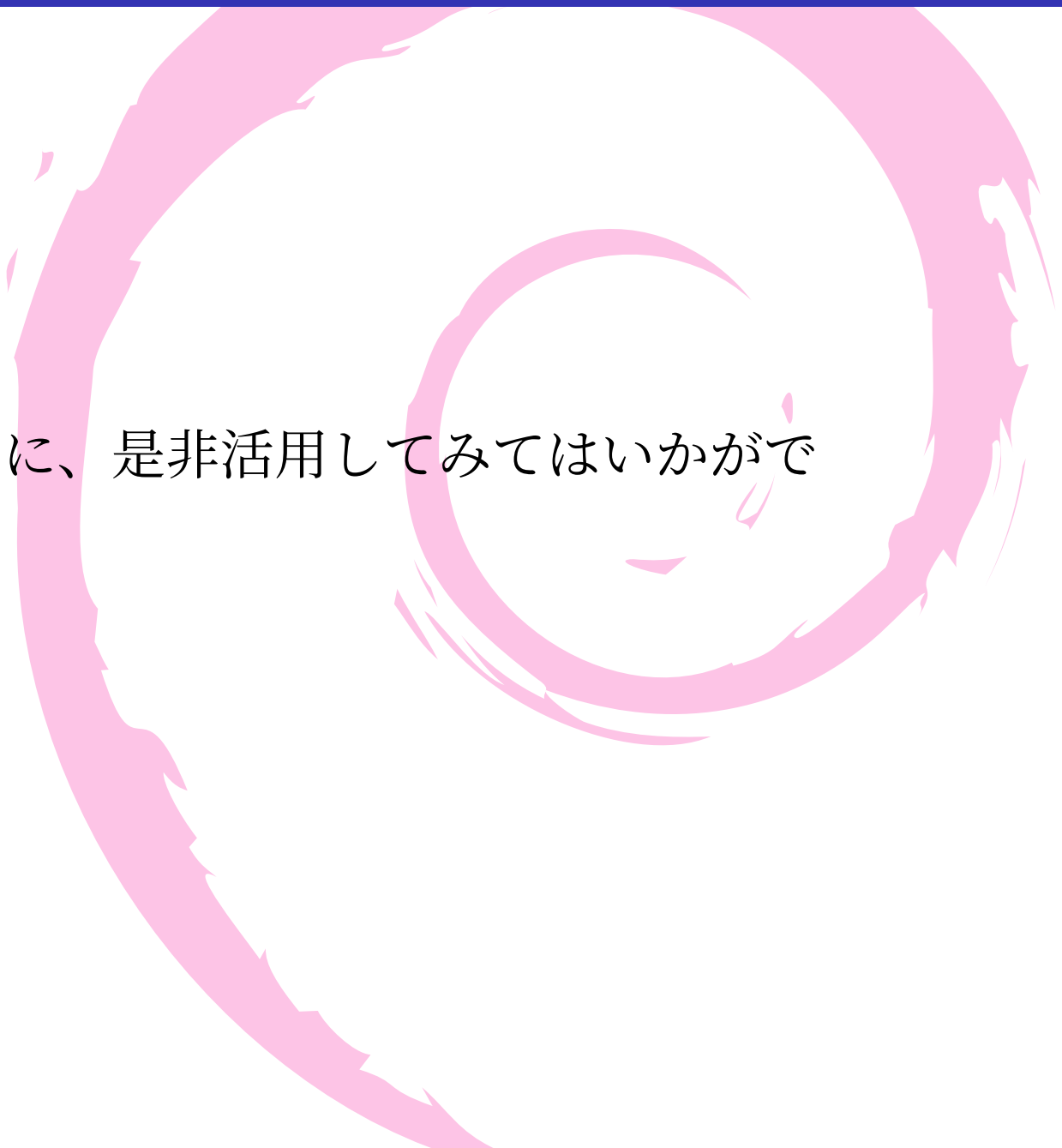
はじめに

Linux カーネルのデバッグ支援機能で人気のあるものに、ftrace という仕組みがあります。今回は Debian unstable で提供されている linux-4.3.3 を利用して、ftrace を使ってみます。

ftrace とは

Linux カーネル内部の実行トレースを効率的に取れるようにしたデバッグ支援機構となります。カーネル内部の C の関数単位でトレースが取れたりします。
人気があるせいか、機能拡張がどんどん続いています。最近では、ユーザプロセスのトレースも取れる機能が追加される、kprobe との連携などが追加されています。

ftrace とは（つづき）



Debian のバグ潰しに、是非活用してみてはいかがでしょう！

ftrace 注意点

評価中に何度か経験したのですが、一部の ftrace の機能 (uevent,kprobe) にて、プローブの手続きを書き間違えたりすると、カーネル全体が突然ハングアップしたり、マシンが突然リブートすることがありました。

　利用にあたっては、必ず十分に事前に動作を検証してから、重要なサーバ上などで実行する事をおすすめします。もちろん、重要でないサーバ上なら、いくらでも試行錯誤くださいませ（笑）

よく知られている使い方について

カーネル内部の関数の実行トレースを取ってみます。なお、実行にあたっては root 権限が必要。

```
# cd /sys/kernel/debug/tracing
# echo function > current_tracer
# head -15 trace
  もしくは、
# cat trace_pipe
```

よく知られている使い方について（つづき）

結果：

```
# tracer: function
#
# entries-in-buffer/entries-written: 205013/84652723   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
     ksoftirqd/2-18    [002] ..s. 34273.981969: _raw_read_lock_bh
                                                     <-ppp_input
     ksoftirqd/2-18    [002] ..s. 34273.981969: _raw_spin_lock_bh
                                                     <-ppp_input
     ksoftirqd/2-18    [002] ..s. 34273.981969: ppp_receive_frame
                                                     <-ppp_input
     ksoftirqd/2-18    [002] ..s. 34273.981969: ppp_receive_nonmp_frame
                                                      <-ppp_input
```

※紙面の都合で、一部改行を入れています。 また、<-XXX は XXX が呼び出し元。

よく知られている使い方について（つづき）

カーネル内部の特定の関数の呼び出しをトレースしてみます。以下のコマンドラインは先程の続きの操作からとなります。

```
一旦、トレースを止める。
# echo nop > current_tracer
schedule 関数の呼び出しのみ取る。
# echo schedule > set_ftrace_filter
 再開してみる
# echo function > current_tracer
# head -15 trace
  もしくは、
# cat trace_pipe
```

※ トレースを止めるには、tracing_on ファイルに 0 や 1 を書き込んで止めるのが本式らしいのですが、今回は簡易に nop を指定して止めています。

よく知られている使い方について（つづき）

結果：

```
# tracer: function
#
# entries-in-buffer/entries-written: 205173/1017318   #P:4
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID   CPU#   ||||    TIMESTAMP  FUNCTION
#              | |       |    ||||       |         |
          <idle>-0     [001] .N.. 35985.931338: schedule
                                         <-schedule_preempt_disabled
        threaded-ml-30937 [001] .... 35985.931365: schedule
                                 <-schedule_hrtimeout_range_clock.part.23
          <idle>-0     [001] .N.. 35985.935088: schedule
                                 <-schedule_preempt_disabled
     kworker/1:1-30635 [001] .... 35985.935118: schedule
                                 <-worker_thread
          <idle>-0     [001] .N.. 35985.937301: schedule
                                 <-schedule_preempt_disabled
```

※紙面の都合で、一部改行を入れています。 また、<-XXX は XXX が呼び出し元。

よく知られている使い方について（つづき）

カーネル内部の特定の関数の呼び出しをトレースしてみます。以下のコマンドラインは先程の続きの操作からとなります。

一旦、トレースを止める。

```
# echo nop > current_tracer
```

schedule 関数の呼び出しのみ取る。

```
# echo schedule > set_ftrace_filter
```

再開してみる

```
# echo function > current_tracer
# head -15 trace
```

もしくは、

```
# cat trace_pipe
```

よく知られている使い方について（つづき）

補足:

- set_ftrace_filter には、*のワイルドカードが利用でき、例えば、
  - `echo '*lock*' > set_ftrace_filter`
  - `echo 'ppp*' > set_ftrace_filter`

  という指定が可能です。*lock*は lock を名前に含む関数がトレースされるようになります。ppp*は行頭が ppp で始まる関数がトレースされるようになります。
- トレースから除外する関数を指定する場合は、set_ftrace_notrace に指定すると合致したものがトレースから除外されます。

よく知られている使い方について（つづき）

参考：set_ftrace_filter に指定可能な関数一覧

```
#  lv available_filter_functions
run_init_process
try_to_run_init_process
do_one_initcall
match_dev_by_uuid
name_to_dev_t
rootfs_mount
... 中略（相当量ある）...
```

よく知られている使い方について（つづき）

コールツリーをとってみます。以下のコマンドラインは先程の続きの操作からとなります。

```
# echo nop > current_tracer
フィルタを一旦クリア
# echo > set_ftrace_filter
# echo do_sys_open > set_graph_function
# echo  function_graph > current_tracer
# head -15 trace
  もしくは、
# tail -f trace
```

よく知られている使い方について（つづき）

結果：

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 2)               |  do_sys_open() {
 2)               |    getname() {
 2)               |      getname_flags() {
 2)               |        kmem_cache_alloc() {
 2)   0.151 us    |          _cond_resched();
 2)   1.725 us    |        }
 2)               |        __do_page_fault() {
 2)   0.194 us    |          down_read_trylock();
 2)   0.090 us    |          _cond_resched();
 2)               |          find_vma() {
 2)   0.191 us    |            vmacache_find();
```

よく知られている使い方について（つづき）

- 他にも、割り込みが長時間止められた関数のトレース (irqsoff) がありますが、Debian の標準の linux-image パッケージでは有効にはなっていません。
  `lv /boot/config-‘uname -r‘` で現在稼働中の linux カーネルのビルド時の設定が確認できます。
  `# CONFIG_IRQSOFF_TRACER is not set` や、
  `# CONFIG_SCHED_TRACER is not set` とあるかと思います。
- 他に利用できるトレースは、`available_tracers` ファイルを確認するとよいと思います。

よく知られている使い方について（つづき）

今回紹介した以上のよく知られている使い方のチュートリアルは、LWN の記事が良いです。

- Ftrace: The hidden light switch
  `https://lwn.net/Articles/608497/`
- Debugging the kernel using Ftrace
  `https://lwn.net/Articles/365835/`
  `https://lwn.net/Articles/366796/`
- Secrets of the Ftrace function tracer
  `https://lwn.net/Articles/370423/`

ユーザプロセスのトレースをやってみる

最近のカーネルの ftrace は、"use user-level statically defined tracing (USDT) probe" というユーザプロセス側のトレースを取れる機能があります。まずはこちらを紹介します。
実際には、uprobe_events ファイルを操作するのですが、関数のエントリポイントのメモリアドレスを指定しなければならないなどの使い勝手の問題があるため、perf-tool というツール経由で操作する方法について述べます。

ユーザプロセスのトレースをやってみる（つづき）

ところで、Debian unstable には、この perf-tool がパッケージ化されているのですが、残念ながら 2015 年 1 月時点の upstream のソースであるため、USDT probe に対応したコマンドが入っていません[1]。仕方が無いので、upstream からソースを引っ張ってくることにします。なお、シェルスクリプトですので、手元に持ってくるだけで使えます。

[1] bug report 上げておきます。

ユーザプロセスのトレースをやってみる（つづき）

ソースの入手と準備

```
$ git clone https://github.com/brendangregg/perf-tools.git
$ cd perf-tools/bin
$ su root
#
```

ユーザプロセスのトレースをやってみる（つづき）

試しに、Debian unstable 全体のプロセスを対象に、libc 中の open 関数を呼び出しをファイル名も含めてトレースしてみます。

```
# ./uprobe 'p:/lib/x86_64-linux-gnu/libc-2.21.so:open
+0(%di):string'
gkrellm-2283  [001] d... 32718.873326: open:
                    (0x7f5543ecf640) arg1="/proc/loadavg"
gkrellm-2283  [001] d... 32718.969105: open:
                    (0x7f5543ecf640) arg1="/proc/loadavg"
gkrellm-2283  [001] d... 32719.064968: open:
                    (0x7f5543ecf640) arg1="/proc/loadavg"
        .... 中略...
```

※紙面の都合で改行を入れています。

ユーザプロセスのトレースをやってみる（つづき）

uprobe の引数の文言についての説明は、linux ソースの `Documentation/trace/uprobetracer.txt` にその説明があります。
先のページの例は、
`/lib/x86_64-linux-gnu/libc-2.21.so` に含まれる open 関数について、関数の第一引数の文字列が格納されている場所の指定（CPU の di レジスタが指し示す先のオフセット 0 番地を文字列データという指定で）[2] を表示せよという意味となります。

[2] AMD64 アーキテクチャでは正確には、edi レジスタですが、現状では di と指定するようです。

ユーザプロセスのトレースをやってみる（つづき）

uprobe はトレース条件も指定でき、例えば、/var/log/以下のファイルを open したプロセスのみをトレース表示せよという指定は、

```
# ./uprobe 'p:/lib/x86_64-linux-gnu/libc-2.21.so:open
  file=+0(%di):string' 'file ~ /var/log/*'
```

となります。

kprobe経由のftraceを使ってみる

linux 3系列のカーネルであれば、ftraceはkprobe機能を活用したトレースを取ることが出来ます。トレース条件にkprobeで指定できるような条件を指定できるので、kprobeに造形が深い場合は、こちらがよいかもしれません。
ここでは、先ほど入手したperf-toolに梱包されているkprobeを利用してみます。

kprobe経由のftraceを使ってみる

使ってみる：

```
# ./kprobe 'p:myopen do_sys_open
        filename=+0(%si):string'
gkrellm-2283  [000] d...   511.849329: myopen:
  (do_sys_open+0x0/0x220) filename="/proc/loadavg"
gkrellm-2283  [000] d...   511.945166: myopen:
  (do_sys_open+0x0/0x220) filename="/proc/loadavg"
gkrellm-2283  [000] d...   512.040938: myopen:
  (do_sys_open+0x0/0x220) filename="/proc/loadavg"
... 中略...
※紙面の都合で改行を入れています。
```

ftrace についてさらに詳しく知る

他にも最新カーネルにはいくつもここでは語られない機能が ftrace に実装されています。さらに詳しく知りたい方は Documentation/trace/以下のファイルを読むとよいと思います。

おわりに

Linux カーネルの ftrace 機能はいろいろと強力です。今回紹介出来なかった、機能もまだまだ存在します。将来、Linux カーネルの挙動を追いかける必要が出てきた時に、ftrace で出来ることをささっと調査できるようになっていると、何かと便利かと思います。

Hack time

成果を記入下さい!

今回、Hack time 時の成果を記録に残してみます。
https://debianmeeting.titanpad.com/7
に皆さんアクセス頂き（認証不要です）、各自成果を 18:20 までに記録するようにおねがいします。

今後のイベント

今後のイベント

- 関西エリア Debian 勉強会。
- 2/13 (土) 第 136 回東京エリア Debian 勉強会
  場所は mkohei さんにてご紹介いただいた場所にて！

今日の宴会場所

今日の宴会場所

未定