第20回FreeBSDワークショップ

佐藤　広生 <hrs@FreeBSD.org>

東京工業大学/ FreeBSD Project

2016/8/1

開催背景

- **日本国内の＊BSDユーザ活動を活発化させましょう**
  - 月1回、東京近辺で定期的な会合を。
  - 講演を聞くだけでなく、話を持ち寄って双方向に議論しましょう

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

本ワークショップの進行

- 19:00〜19:30　自己紹介＋話題にしたいトピック
- 19:30〜20:00　ライトニングトーク
- 20:00〜20:10　休憩
- 20:10〜21:15　DTrace 入門

意見は自由に発言ください！

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

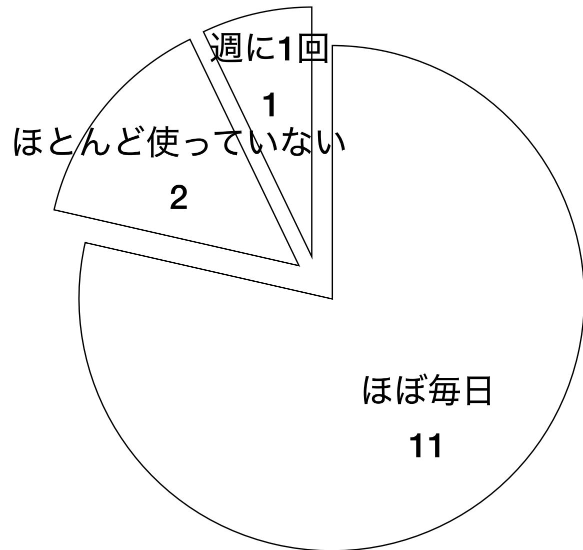
オーガナイザの自己紹介

- 名前：佐藤 広生
  - FreeBSD コアチームメンバ、リリースエンジニア(2006-)
  - FreeBSD Foundation 理事(2008-)
  - その他の＊BSD／オープンソース関連の活動いろいろ
  - 東京工業大学助教(2009-)

自己紹介タイム

- 名前（所属）
- 開発者 or 利用者
- 興味がある／話題にしたい内容

をどうぞ



今回の出席者内訳：新規2名、再参加者12名

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

メモ

メモ

https://people.allbsd.org/~hrs/sato-FBSDW20160715.pdf
https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

DTrace入門（続き）

佐藤　広生 <hrs@FreeBSD.org>

東京工業大学/ FreeBSD Project

2016/8/1

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

お話しすること

- DTraceで性能測定
- ユーザランドプログラムで使うには

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

復習：DTraceとは？

- システムやアプリケーションプログラムの
  内部動作を稼働中に調べるためのツール
  - 動作を調べる
  - ベンチマークを取る
  - などなど。

- **「稼働中に」「対象を変更せずに」**がポイント

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

復習：使い方

- 前準備は、前の資料の「準備」を参照のこと

- 例：今走っているプロセスが 10秒間でread(2)のシステムコールを何回発行し、どれくらいのデータ長のデータを読んだのか調べたい。

```
# dtrace -c "sleep 10" -n '
syscall::read:entry /execname != "dtrace"/
{ @reads[execname, arg2] = count(); }'
```

復習：使い方

- 前準備は、前の資料の「準備」を参照のこと

- 例：今走っているプロセスが 10秒間でread(2)のシステムコールを何回発行し、どれくらいのデータ長のデータを読んだのか調べたい。

プローブ　　述語

```
syscall::read:entry /execname != "dtrace"/
{
    @reads[execname, arg2] = count();
}
```

アクション

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

復習：使い方

- 例：今走っているプロセスが 10秒間でread(2)のシステムコールを何回発行し、どれくらいのデータ長のデータを読んだのか調べたい。

```
# dtrace -n 'syscall::read:entry /execname != "dtrace"/ { @reads[execname, arg2] = count(); }'
-c "sleep 10"
dtrace: description 'syscall::read:entry ' matched 2 probes
dtrace: pid 98661 has exited

  ftpd                          71              1
  ftpd                         128              1
  ftpd                         260              1
  rsync                     262144              1
  ftpd                       41448              2
  ftpd                     1048600              2
  sshd                       16384              2
  inetd                        260              3
  inetd                      32768              9
  ftpd                       32768             14
  spegla                      4096             16
  spegla                     16384             18
  cvsupd                      8192            203
  cvsupd                      4096           1249
  spegla                         1           1653
```

プロセス名　arg2　回数

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

性能測定

- I/Oの測定：ioの発生原因とデータ長を出す　ioプロバイダ

```
profile:::tick-5s
{
    exit(0);
}
io:::start /args[0] != NULL/
{
    @[pid, execname] = quantize(args[0]->bio_bcount);
}
```

5秒で終わり

IOのデータ長

```
  20976  sync
         value  ------------- Distribution ------------- count
          1024 |                                         0
          2048 |@@@@@@@@@@@@@@@@@@@@@@@@@@@              2
          4096 |                                         0
          8192 |                                         0
         16384 |@@@@@@@@@@@@@                            1
         32768 |                                         0
```


https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

性能測定

- I/Oの測定：特定のプロセスのioの発生原因とデータ長を出す

```
profile:::tick-5s
{
    exit(0);
}
io:::start /pid == $target && args[0] != NULL/
{
    @[ustack()] = count();
}
```

$target は「-c コマンド」もしくは -p PIDで指定

ustack()は、ユーザランドのスタックトレースを返す

性能測定

- I/Oの測定：特定のプロセスのioの発生原因とデータ長を出す

```
# dtrace -s iopid.d -c 'cp -R /var/log /tmp'
dtrace: script 'iopid.d' matched 2 probes
dtrace: pid 21004 has exited
            libc.so.7`__sys_getdirentries+0x7
              0x3
              0xc985c031
                1
             libc.so.7`mkdir+0x7
              cp`0x804c6d8
              0x585
                1
             libc.so.7`__sys_openat+0x7
              0xffffff9c
              0x9090c35d
                1
             libc.so.7`_write+0x7
              0x4
              0x44f8e0f
              177
```


https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

性能測定

- I/Oの測定：I/Oのレイテンシをデバイス単位で出す

```
#!/usr/sbin/dtrace -s
#pragma D option quiet
io:::start
{
        start_time[arg0] = timestamp;
}

io:::done /this->start = start_time[arg0]/
{
        this->delta = (timestamp - this->start) / 1000;
        @a[args[1]->device_name, args[1]->unit_number] =
            quantize(this->delta);
        start_time[arg0] = 0;
}

profile:::tick-10s
{
        exit(0);
}
```

#pragma D option quiet は「-q」と同じ

timestampは現在時刻。

io-startで代入

io-doneで代入

デバイス名(da0)など

性能測定

- I/Oの測定：I/Oのレイテンシをデバイス単位で出す

```
da0
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |                                         1
               2 |                                         0
               4 |                                         2
               8 |                                         2
              16 |                                         1
              32 |                                         13
              64 |@                                        78
             128 |@@@@@@@@@@@@@@@@@@@@@@@@@@@              3971
             256 |@@@@@@@@                                 1125
             512 |@@@                                      490
            1024 |@@                                       232
            2048 |                                         65
            4096 |                                         11
            8192 |                                         0
```


性能測定

- ネットワーク：
  80/tcpに到着するパケット数をクライアントIP別に集計

tcpプロバイダ

tcp:::receiveはTCP接続を受け取った時点

```
tcp:::receive /args[4]->tcp_dport == 80/
{
  @[args[2]->ip_daddr] = count();
}

# dtrace -s httpc.d -c "sleep 10"
dtrace: script 'httpc.d' matched 1 probe
dtrace: pid 21183 has exited

  127.0.0.1                                                        12
  ::1                                                              12
```

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

性能測定

- ネットワーク： TCP接続にかかる時間を測定

cs_cidは接続ID

```
tcp:::connect-request
{
        start[args[1]->cs_cid] = timestamp;
}
tcp:::connect-established /start[args[1]->cs_cid]/
{
        @["Latency (us)", args[2]->ip_daddr] =
            quantize(timestamp - start[args[1]->cs_cid]);
        start[args[1]->cs_cid] = 0;
}
# dtrace -s tcplatency.d -c "nc -z www.yahoo.com 80"
dtrace: script 'tcplatency.d' matched 2 probes
Connection to www.yahoo.com 80 port [tcp/http] succeeded!
dtrace: pid 21350 has exited

  Latency (us)                                        172.16.87.129
           value  ------------- Distribution ------------- count
          131072 |                                         0
          262144 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
          524288 |                                         0
```

ncを使って接続

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

性能測定

- ネットワーク： TCPステートマシンの遷移動作を見る

```
#pragma D option quiet
#pragma D option switchrate=10

dtrace:::BEGIN
{
        printf(" %12s %3s %12s  %-20s    %-20s\n",
            "PID", "CPU", "DELTA(us)", "OLD", "NEW");
        last = timestamp;
}

tcp:::state-change
{
        this->elapsed = (timestamp - last) / 1000;
        printf(" %12d %3d %12d  %-20s -> %-20s\n",
            pid, cpu, this->elapsed,
            tcp_state_string[args[5]->tcps_state],
            tcp_state_string[args[3]->tcps_state]);
        last = timestamp;
}
```

最初に一回

tcps_stateはステートを保持している

性能測定

- ネットワーク： TCPステートマシンの遷移動作を見る

wgetを使って接続：TCP接続が2本出てる！

```
# dtrace -s tcpstate.d -c "wget -o /dev/null http://www.google.com"

      PID CPU    DELTA(us)  OLD                      NEW
    21473   1       350651  state-closed         -> state-syn-sent
       12   1       103086  state-syn-sent       -> state-established
    21473   1        64244  state-closed         -> state-syn-sent
       12   1       102376  state-syn-sent       -> state-established
    21473   1        78939  state-established    -> state-fin-wait-1
       12   1          187  state-fin-wait-1     -> state-fin-wait-2
       12   1         1544  state-fin-wait-2     -> state-time-wait
    21473   1        16546  state-established    -> state-fin-wait-1
       12   1          729  state-fin-wait-1     -> state-fin-wait-2
       12   1           16  state-fin-wait-2     -> state-time-wait
```

PID=12ってなんだ？

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

ユーザランドDTrace

- PIDプロバイダを使ってユーザランドも調べられる
- そもそもDTraceの動作は、カーネル・ユーザランドの区別を特別扱いしない
- ユーザランド関数のプローブ：

```
pid$target:procname:probefunc:entry
```

ユーザランドDTrace

- dtrace -l を見てみる

```
# dtrace -l -P pid\$target -c "/bin/ls" | wc -l
230383
```

- 「pid$target:共有ライブラリ名:関数名:プローブ名」

```
# dtrace -l -P pid\$target -c "/bin/ls"
....
56421   pid21553  libutil.so.9  forkpty return
....
269515  pid21553  libc.so.7    poll entry
.....
```


https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

ユーザランドDTrace

- /bin/echoを実行して、libc の関数を呼んでるところを見る

```
# dtrace -n 'pid\$target:libc*::entry { @[probefunc] = count(); }' \
        -c "/bin/echo"
dtrace: description 'pid\$target:libc*::entry ' matched 2996 probes

dtrace: pid 21579 has exited

  __malloc                                                          1
  __sys_exit                                                        1
  __sys_writev                                                      1
  _exit                                                             1
  _writev                                                           1
  atexit                                                            1
  exit                                                              1
  malloc                                                            1
  memset                                                            1
  mmap                                                              1
  writev                                                            1
  __cxa_finalize                                                    2
```

「pid$target:**libc***::entry」を見ている

ユーザランドDTrace

- 自分で作ったプログラムならどうなの？

```
#include <stdio.h>

void
myfunc(int i)
{
        printf("...%d\n", i);
        return;
}

int
main(void)
{
        printf("hello, world\n");
        myfunc(100);

        return(0);
}
```


https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

ユーザランドDTrace

- 自分で作ったプログラムならどうなの？

```
# cc hello.c
# dtrace -q -n 'pid\$target:a.out:myfunc:entry    \
             { printf("entry = %d\n",arg0); }' \
             -c ./a.out
hello, world
...100
entry = 100
#
```

「pid$target:実行ファイル名:関数名:entry」になるところに注意

```
# dtrace -l -m pid\$target:a.out -c ./a.out
53695   pid21805            a.out                           _start entry
53696   pid21805            a.out                           myfunc return
53697   pid21805            a.out                           myfunc entry
53698   pid21805            a.out                           myfunc 0
53699   pid21805            a.out                           myfunc 1
....
```

「-m プロバイダ:モジュール」で表示を制限できる

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

ユーザランドDTrace

- MySQLやPostgreSQLは、DTraceに対応している
  → mysql$target:::filesort-start や
  mysql$target:::filesort-done が定義されていて、
  データベースの検索の性能分析や動作をトレースできる

```
mysql$target:::filesort-start
{
        self->ts = timestamp;
        printf("Sort start: %s", copyinstr(arg0));
}
mysql$target:::filesort-done
{
        printf("Sort done: %d ms / Result: %s",
            (timestamp - self->ts) / 1000000,
            copyinstr(arg0));
}
```


https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

ユーザランドDTrace

- DTraceに対応してるという意味は？
  → 独自のプロバイダを持っているということ
  （SDT: Statically Defined Tracingというタイプのプローブ）
- 関数単位であれば、対応していなくても使える
- USDTの作り方：

```
% cat provider.d
provider database {
        probe query__start(char *);
        probe query__done(char *);
};

% dtrace -h -s provider.d

DATABASE_QUERY_START("hoge") -> database$target:::query-start
DATABASE_QUERY_DONE("fuga")  -> database$target:::query-done
```

定義

provider.hの生成

対応関係

ユーザランドDTrace

```
#include <stdio.h>

#include <sys/sdt.h>
#include "provider.h"

int
main(void)
{
        /* Give us time to start DTrace */
        sleep(5);

        DATABASE_QUERY_START("SELECT * FROM apples");
        /* simulate a long query */
        sleep(1);
        DATABASE_QUERY_DONE("TOO MANY APPLES");

        return (0);
}
```

provider.h とsys/sdt.h をinclude

定義したものを使う

注意：libelf をリンクすること

まとめ

- 一見複雑そうに見える測定でも、数行のスクリプトでいけるだけの記述力
- デバッグ、ユニットテスト、動作の検証、性能測定など、いろいろな用途に重宝するはず
- 触ってみましょう！

https://people.allbsd.org/~hrs/sato-FBSDW20160801.pdf

おしまい

- 質問はありますか？

告知

- FreeBSDワークショップ（ほぼ月一回）
  （次回は9月の中旬？）

- AsiaBSDCon 2017
  2017/3/9-12
- 東京理科大学 森戸記念館
  飯田橋駅から徒歩5分、東京理科大学の施設