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

第19回FreeBSDワークショップ

佐藤　広生 <hrs@FreeBSD.org>

東京工業大学/ FreeBSD Project

2016/7/15

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

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-FBSDW20160715.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-FBSDW20160715.pdf

オーガナイザの自己紹介

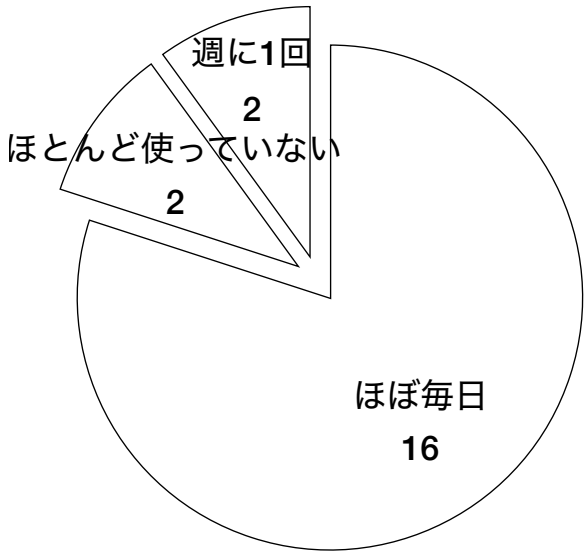
- 名前：佐藤 広生
  - FreeBSD コアチームメンバ、リリースエンジニア(2006-)
  - FreeBSD Foundation 理事(2008-)
  - その他の＊BSD／オープンソース関連の活動いろいろ
  - 東京工業大学助教(2009-)

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

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5名、再参加者15名

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

メモ

メモ

DTrace入門

佐藤　広生 <hrs@FreeBSD.org>

東京工業大学/ FreeBSD Project

2016/7/15

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

お話しすること

- 使ってみよう DTrace
- DTraceとは一体何なのか？
- DTraceでできること

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

DTraceとは？

- 名前は聞いたことがあるけどよく知らない...
- 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
- 2005年3月にSolarisに追加され、現在は
  OpenSolaris(illumos), Mac OSX, FreeBSD, NetBSD, Linux
  などで使える（カーネルが対応している必要あり）

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

DTraceとは？

- Unix系OSでシステムの動作を調べるコマンドはたくさんある
- **Solaris:** sar(1), vmstat(1M), mpstat(1M), iostat(1M), netstat(1M), kstat(1M), prstat(1M),...
- **Mac OSX:** sar(1), vm_stat(1), top(1),...
- **FreeBSD:** systat(1), vmstat(8), iostat(8), netstat(8), sockstat(1), procstat(1), top(1),...
- DTrace は、これらと何か違うものなのか？

DTraceの特徴

- 個々のツールは統計情報の取得にCPU時間を多く使う
  → top(1)はとても重いユーティリティです。
- ktraceやtrussのようなツールはプロセス単位でしか
  情報が取れない
- システムコールなどのカーネルの動作は調べられるけれど、
  ユーザランドプログラムの動作を調べる方法は少ない
- 知りたい情報以外もたくさん出てきてしまう

- 例：今あるプロセスが 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！

準備

- FreeBSDの動作を調べるには

- FreeBSD 10 系ならば GENERIC カーネルで使えます。
- 9は
  options KDTRACE_HOOKS
  options DDB_CTF
  options KDTRACE_FRAME
  makeoptions DEBUG="-g"
  makeoptions WITH_CTF=1
  を入れてカーネルを作る必要があります。
  （ハンドブック参照のこと）

準備

- FreeBSD以外のユーザランドプログラムにも
  DTraceを使いたい場合には
  （今回の話では詳細まで行きません）

- /etc/make.confに
  STRIP=
  CFLAGS+=-fno-omit-frame-pointer
  WITH_CTF=1
  を付けてください

準備

- DTraceカーネルモジュールを読む

```
# kldload dtraceall
```

- dtraceコマンドを実行する

```
# dtrace -l
```

- 覚えておこう
  - 基本的にdtraceコマンドだけを使う
  - root権限が必要
  - カーネルモジュールは、読み込まれていなければ
    dtraceコマンドを実行する時点で自動で読み込まれる

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

準備

```
# dtrace -l
  ID   PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                      BEGIN
    2     dtrace                      END
    3     dtrace                      ERROR
    4        fbt            kernel                camstatusentrycomp entry
    5        fbt            kernel                camstatusentrycomp return
    6        fbt            kernel             cam_compat_handle_0x17 entry
    7        fbt            kernel             cam_compat_handle_0x17 return
:
:
:
```

- -l を指定して実行すると、何かたくさん行が出てくる

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

使ってみよう

- 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。

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

使ってみよう

- 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(); }'
```


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

使ってみよう

- 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
```

使ってみよう

- 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
```


何をやったのか？

- dtraceコマンドは、D言語（注意：Walter Bright のD言語ではなくDTrace専用の簡易言語）で書かれたスクリプトを実行する
- 「-n "スクリプト本体"」、もしくは「-s スクリプトファイル名」
- 「-c "コマンド"」は、指定したコマンドが終了したら dtrace を止めるという指定

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

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

何をやったのか？

- D言語
  - AWKにとても良く似た文法
  - 「プローブ /述語/ { 手続き }」という宣言を並べるだけ！

- プローブ：調べたいポイント。「-l」で出てきたリストがそれ。
  - `syscall::read:entry` = `read(2)`が呼ばれた時点

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


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

何をやったのか？

- D言語
  - AWKにとても良く似た文法
  - 「プローブ /述語/ { 手続き }」という宣言を並べるだけ！

- 述語：条件の絞り込み（省略しても良い）
  - `/execname != "dtrace"/`
    → 実行ファイル名が `dtrace`でないもの

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

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

何をやったのか？

- D言語
  - AWKにとても良く似た文法
  - 「プローブ /述語/ { 手続き }」という宣言を並べるだけ！
- 手続き：何をやるか
  - `execname` と `arg2` をキーにした連想配列に回数をセット

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


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

詳しく読んでみる

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

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

詳しく読んでみる

プローブはread(2)の入り口

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

- プローブを知るには？
  - dtrace -l で一覧が表示される
  - 「プロバイダ：モジュール：関数：プローブ名」
  - システムコールなら、syscallプロバイダにある
  - 省略や「*」が使える（syscall::read*: とするとreadvも対象）

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

詳しく読んでみる

述語


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


- 述語で使える書き方
  - AWKと同じ（regexではない）。組み込み変数が使える。

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

DTraceの組み込み変数

名前	意味
arg0, arg1, ... arg9	引数。int64_t。定義されていなければ0になる
args[]	引数。型は定義されたもの。
cpu	実行しているCPUのID
curpsinfo	現在のプロセスの情報（struct psinfo_t）
errno	最後のシステムコールのerrno
execname	現在のプロセス名
pid, ppid, uid, tid	PID, PPID, UID, スレッドID
timestamp	時刻

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

DTraceの組み込み変数

名前	意味
arg0, arg1, ... arg9	引数。int64_t。定義されていなければ0になる
args[]	引数。型は定義されたもの。
cpu	実行しているCPUのID
curpsinfo	現在のプロセスの情報（構造体）
curthread	現在のスレッドの情報（構造体）
errno	最後のシステムコールのerrno
execname	現在のプロセス名
pid, ppid, uid, tid	PID, PPID, UID, スレッドID
timestamp	時刻

- ここで出てくる構造体の定義は/usr/lib/dtrace/にある。
- カーネルのヘッダファイルにある構造体ももちろん使用可能

```
typedef struct psinfo {
        int     pr_nlwp;        /* number of threads */
        pid_t   pr_pid;         /* unique process id */
        pid_t   pr_ppid;        /* process id of parent */
        pid_t   pr_pgid;        /* pid of process group leader */
        pid_t   pr_sid;         /* session id */
        uid_t   pr_uid;         /* real user id */
        uid_t   pr_euid;        /* effective user id */
        gid_t   pr_gid;         /* real group id */
        gid_t   pr_egid;        /* effective group id */
        uintptr_t
                pr_addr;        /* address of process */
        string  pr_psargs;      /* process arguments */
        u_int   pr_arglen;      /* process argument length */
        u_int   pr_jailid;      /* jail id */
} psinfo_t;
```

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

詳しく読んでみる

述語

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

- 述語で使える書き方
  - AWKと同じ（regexではない）。組み込み変数が使える。
  - プロセス名が dtrace と一致しないもの全部、という意味。

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

詳しく読んでみる

述語

```
syscall::read:entry
/execname != "dtrace" && uid == 1000/
{
     @reads[execname, arg2] = count();
}
```

- 述語で使える書き方
  - AWKと同じ（regexではない）。組み込み変数が使える。
  - プロセス名が dtrace と一致しないもの全部、という意味。
  - こうすると、さらに UID が 1000 に一致するものに限定

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

詳しく読んでみる

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

- 手続きで使える書き方
  - これもAWKにとても似ている
  - 変数は宣言せずに使える。C言語風のキャスト文法にも対応
  - 宣言もできる。char, short(int16_t), int(int32_t), long long(int64_t), float, double, long doubleが使える。

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

詳しく読んでみる

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

- read(2) のarg2って何だ？

```
% man 2 read

                                    ↓ここ
ssize_t
read(int fd, void *buf, size_t nbytes);
```

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

詳しく読んでみる

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

- 手続きで使える書き方
  - 文字列型として string がある。
  - []をつけると配列型になり、連想配列になる。配列のキーは"," で区切って複数指定可能。
  - 構造体も定義可能。/usr/lib/dtrace/*.d の内容がデフォルト

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

詳しく読んでみる

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

- 手続きで使える書き方
  - スレッドローカル変数として、self->x = 1 という書き方ができる。0 を代入するか、スレッドがなくなると解放される。
  - 手続きローカル変数として、this->x = 1 という書き方ができる。

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

詳しく読んでみる

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

- 集約変数 @ と集約関数
  - カウンタ用の特殊文法。
  - @a = count(); とすると、呼ばれる度に a++ される。
  - @a[execname] = count(); とすると、プロセス名単位に
- a++ と同じだが、SMP環境での挙動がポイント

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

詳しく読んでみる

- 集約変数 @ と集約関数
  - 集約変数の代入は集約関数しかできない
  - count()：カウンタの値を1増やす
  - sum(a)：カウンタの値をa増やす
  - avg(a)：カウンタの平均値を計算する
  - min(a), max(a)：カウンタに最小値 or 最大値を保持する
  - quantize(a)：2の冪乗の単位でヒストグラムを生成する
  - lquantize(a, b, c, d)：線形なヒストグラムを生成する

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

もっと使ってみよう

- 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(); }'
```


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

もっと使ってみよう

- 例：**rsyncのプロセス**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て**ヒストグラム**にしたい

```
# dtrace -c "sleep 10" -n '
syscall::read:entry /execname == "rsync"/
{ @reads = quantize(arg2); }'
```

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

もっと使ってみよう

- 例：**rsyncのプロセス**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て**ヒストグラム**にしたい

```
# dtrace -c "sleep 10" -n '
syscall::read:entry /execname == "rsync"/ { @reads = quantize(arg2); }'
dtrace: description 'syscall::read:entry ' matched 2 probes
dtrace: pid 14340 has exited


           value  ------------- Distribution ------------- count
           16384 |                                         0
           32768 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 32
           65536 |                                         0
```


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

もっと使ってみよう

- 例：**rsyncのプロセス**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て**ヒストグラム**にしたい

```
# dtrace -c "sleep 10" -n '
syscall::read:entry /execname == "rsync"/ { @reads = quantize(arg2); }'
dtrace: description 'syscall::read:entry ' matched 2 probes
dtrace: pid 14340 has exited
```

32kBのreadが32回

```
           value  ------------- Distribution ------------- count
           16384 |                                         0
           32768 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 32
           65536 |                                         0
```

quantize() は回数と大きさのヒストグラムを生成する

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

もっと使ってみよう

- スクリプトにして実行しよう
  例：**5秒の間に、プロセス名と、そのプロセスが開いた**
  パス名のリストを出すスクリプト

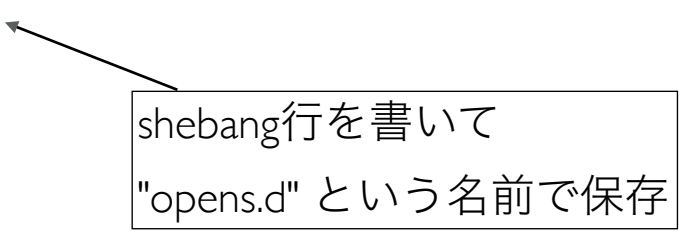
```
#!/usr/sbin/dtrace -s
profile:::tick-5s
{
        exit(0);
}
syscall::open:entry
{
  @opens[execname, copyinstr(arg0)] = count();
}
```


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

もっと使ってみよう

- スクリプトにして実行しよう
  例：**5秒の間に、プロセス名と、そのプロセスが開いた**
  パス名のリストを出すスクリプト

```
#!/usr/sbin/dtrace -s
profile:::tick-5s
{
        exit(0);
}
syscall::open:entry
{
  @opens[execname, copyinstr(arg0)] = count();
}
```



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

もっと使ってみよう

- スクリプトにして実行しよう
  例：**5秒の間に、プロセス名と、そのプロセスが開いた**
  パス名のリストを出すスクリプト

```
#!/usr/sbin/dtrace -s
profile:::tick-5s
{
        exit(0);
}
syscall::open:entry
{
  @opens[execname, copyinstr(arg0)] = count();
}
```

profile:::tick-Ns というプローブは、
N秒後に一回マッチするプローブ

exit()関数は、dtraceを終了させる

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

もっと使ってみよう

- スクリプトにして実行しよう
  例：**5秒の間に、プロセス名と、そのプロセスが開いた**
  パス名のリストを出すスクリプト

```
# ./opens.d
dtrace: script './opens.d' matched 3 probes
CPU     ID                    FUNCTION:NAME
  0  55680                         :tick-5s

  ftpd                /etc/group                          8
  ftpd                /etc/pwd.db                         8
  ftpd                /usr/share/zoneinfo/UTC             8
  ftpd                /usr/share/zoneinfo/posixrules      8
  ftpd                .                                  24
#
```

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

もっと使ってみよう

- 前回のZFSの性能チューニングで使ったスクリプト

```
/*
 * Measure ZFS transaction group statistics
 */
txg-syncing /((dsl_pool_t *)arg0)->dp_spa->spa_name == $$1/
{
        start = timestamp;
        this->dp = (dsl_pool_t *)arg0;
        d_total = this->dp->dp_dirty_total;
        d_max = `zfs_dirty_data_max`;
}
txg-synced /start && ((dsl_pool_t *)arg0)->dp_spa->spa_name
== $$1/
{
        this->d = timestamp - start;
        printf("%4dMB of %4dMB synced in %d.%02d seconds",
        d_total / 1024 / 1024,
        d_max / 1024 / 1024, this->d / 1000000000,
        this->d / 10000000 % 100);
}
```


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

もっと使ってみよう

- 前回のZFSの性能チューニングで使ったスクリプト

```
/*
 * Measure ZFS transaction group statistics
 */
txg-syncing /((dsl_pool_t *)arg0)->dp_spa->spa_name == $$1/
{
        start = timestamp;
        this->dp = (dsl_pool_t *)arg0;
        d_total = this->dp->dp_dirty_total;
        d_max = `zfs_dirty_data_max`;
}
txg-synced /start && ((dsl_pool_t *)arg0)->dp_spa->spa_name
== $$1/
{
        this->d = timestamp - start;
        printf("%4dMB of %4dMB synced in %d.%02d seconds",
        d_total / 1024 / 1024,
        d_max / 1024 / 1024, this->d / 1000000000,
        this->d / 10000000 % 100);
}
```


```
# dtrace -l | grep txg
57913     sdt         zfs                 none txg-quiesced
57914     sdt         zfs                 none txg-quiescing
57915     sdt         zfs                 none txg-opened
57916     sdt         zfs                 none txg-syncing
57917     sdt         zfs                 none txg-synced
```

実際には sdt:zfs:none:txg-syncing という名前。

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

もっと使ってみよう

- 前回のZFSの性能チューニングで使ったスクリプト

```
/*
 * Measure ZFS transaction group statistics
 */
txg-syncing /((dsl_pool_t *)arg0)->dp_spa->spa_name == $$1/
{
        start = timestamp;
        this->dp = (dsl_pool_t *)arg0;
        d_total = this->dp->dp_dirty_total;
        d_max = `zfs_dirty_data_max;
}
txg-synced /start && ((dsl_pool_t *)arg0)->dp_spa->spa_name
  == $$1/
{
        this->d = timestamp - start;
        printf("%4dMB of %4dMB synced in %d.%02d seconds",
        d_total / 1024 / 1024,
        d_max / 1024 / 1024, this->d / 1000000000,
        this->d / 10000000 % 100);
}
```

$$1 は、dtraceコマンドの引数の1番目（ここではzpool名）

このへんはカーネルのソースを読んで、
どのデータにアクセスしているのか調べないとわかりません…

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

もっと使ってみよう

- とてもとても複雑な例

```
# less /usr/sbin/dtruss
```

- truss(1)をD言語で書いたもの。そのまま実行できます。

```
# dtruss df -h
SYSCALL(args)             = return
mmap(0x0, 0x8000, 0x3)            = 6422528 0
issetugid(0x0, 0x0, 0x0)                  = 0 0
lstat("/etc\0", 0x7FFFFFFFD3E8, 0x0)             = 0 0

...
```

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

まとめ

- DTraceとは、カーネルやユーザランドプログラムの挙動を調べるための機構です。
- DTraceを使うには、カーネルが対応する必要があります。FreeBSD, Mac OS X などは、標準インストールで使えます。
- プローブ（データ収集点）とそれに対する処理をD言語で書き、イベントドリブンで動作します。
- D言語はAWKの文法によく似ています。

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

まとめ

- スクリプトを書くにはソースを読む能力が必要ですが、すでにたくさんのスクリプトが公開されています。性能測定用スクリプトなどは、詳しい内部構造を知らなくても使えます。
- プローブの種類は、OSによって異なります。他のOS用のスクリプトを持ってくる時には要注意。
- 既存のコードを変更せず、性能与える影響を最小限に抑えるように設計されているため、デバッグなどにも便利です（開発者はprintfデバッグの代わりに使えて重宝します）

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

さらなる話題

- ネットワークスタックの挙動のトレースなど、
  もっと具体的なスクリプトの書き方の例は？
- 自分の作ったユーザランドプログラムで
  DTraceを使うには?
- 需要があれば、次回にもうちょっと実用的な例について
  お話します。

https://people.allbsd.org/~hrs/sato-FBSDW20160715.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8月1日月曜日）

- 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