Профилирование кода на C/C++ в *nix системах

Александр Алексеев
http://eax.me/

HighLoad++

Профессиональная конференция разработчиков высоконагруженных систем

Два слова о себе

DevZen
PODCAST

В этом толке

- gettimeofday
- strace, ltrace, truss
- gprof
- gdb / lldb
- perf
- pmcstat
- SystemTap
- DTrace
- HeapTrack
- BPF / bcc
- **Умные книжки и красивые картинки!**
- **Важно: не заменяет документацию (via Роман Поборчий)**

Дисклеймер

Disclaimer

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?
- Отсутствие анализа

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?
- Отсутствие анализа
- Игнорирование ошибок

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?
- Отсутствие анализа
- Игнорирование ошибок
- Неправильные настройки

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?
- Отсутствие анализа
- Игнорирование ошибок
- Неправильные настройки
- Нетипичная нагрузка

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?
- Отсутствие анализа
- Игнорирование ошибок
- Неправильные настройки
- Нетипичная нагрузка
- Маркетинг и подгон

Как не нужно делать бенчмарки

Типичные ошибки (список не полон):

- Неповторяемость
- Вы измеряете не то, что думаете
- Взятие среднего
- Кто будет бенчмаркать бенчмарки?
- Отсутствие анализа
- Игнорирование ошибок
- Неправильные настройки
- Нетипичная нагрузка
- Маркетинг и подгон
- Игнорирование других свойств (стоимости и т.д.)

gettimeofday(), пример кода

```
#include <sys/time.h>

/* ... */

struct timeval  tv;
gettimeofday(&tv, NULL);

double time_begin = ((double)tv.tv_sec) * 1000 +
                    ((double)tv.tv_usec) / 1000;

/* ... */

gettimeofday(&tv, NULL);
double time_end = ((double)tv.tv_sec) * 1000 +
                  ((double)tv.tv_usec) / 1000 ;

double total_time_ms = time_end - time_begin;

printf("TOTAL TIME (ms) = %f\n", total_time_ms);
```

gettimeofday(), плюсы/минусы

- Не такой уж дорогой метод, спасибо VDSO
- Но со спинлоками все равно не стоит использовать
- Удобно, если что-то тормозит ***иногда***

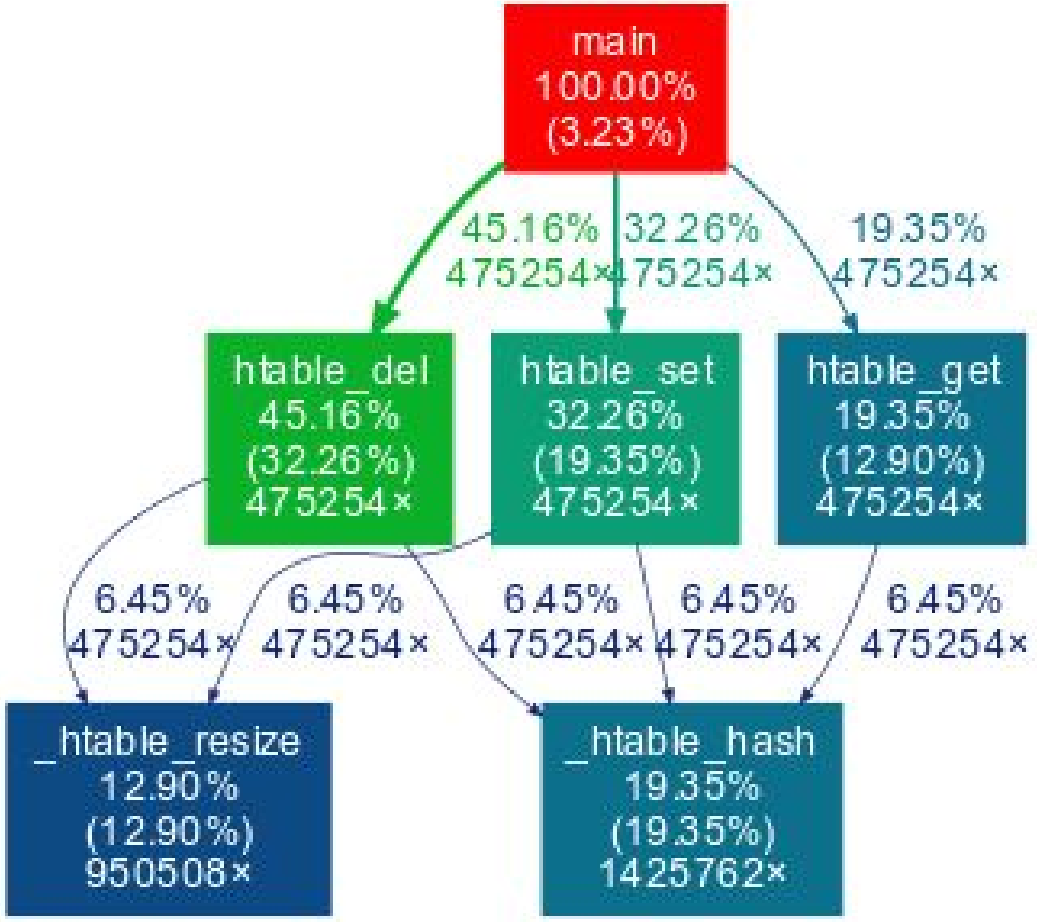
strace, ltrace, truss

```
[eax@e733 ~]$ ltrace -c pwd
/usr/home/eax
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 34.44    0.000083          83         1 puts
 34.02    0.000082          82         1 getcwd
 19.50    0.000047          47         1 atexit
 12.03    0.000029          29         1 getopt
------ ----------- ----------- --------- --------------------
100.00    0.000241                     4 total
[eax@e733 ~]$ truss -c pwd 2>&1 | head
/usr/home/eax
syscall                   seconds   calls  errors
__sysctl               0.000014189       2       0
issetugid              0.000010237       2       0
write                  0.000009069       1       0
sysarch                0.000005182       1       0
sigprocmask            0.000063035      12       0
readlink               0.000008329       1       1
read                   0.000016261       2       0
openat                 0.000024322       3       0
[eax@e733 ~]$
```

gprof, текстовый формат

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 30.37      0.10     0.10   475254   210.86   295.21  htable_del
 18.22      0.16     0.06  1425762    42.17    42.17  _htable_hash
 18.22      0.22     0.06   475254   126.52   210.86  htable_set
 12.15      0.26     0.04   950508    42.17    42.17  _htable_resize
 12.15      0.30     0.04   475254    84.34   126.52  htable_get
  3.04      0.31     0.01                             main
  0.00      0.31     0.00        1     0.00     0.00  htable_free
  0.00      0.31     0.00        1     0.00     0.00  htable_new
```

gprof, построенная картинка



gdb, lldb — получение бэктрейса (bt)

$ gdb --batch --command=gdb.batch -p 12345

$ lldb -p 62510 --batch -s lldb.batch

Полезный прием при поиске и устранении lock contention.

gdb, lldb — реальный патч (44ca4022)

```
commit 44ca4022f3f9297bab5cbffdd97973dbba1879ed
Author: Robert Haas <rhaas@postgresql.org>
Date:   Wed Mar 23 10:56:23 2016 -0400

    Partition the freelist for shared dynahash tables.

    Without this, contention on the freelist can become a pretty serious
    problem on large servers.

    Aleksander Alekseev, reviewed by Anastasia Lubennikova, Dilip Kumar,
    and me.
```

http://habr.ru/p/310372/ (Примеры реальных патчей в PostgreSQL, часть 2)

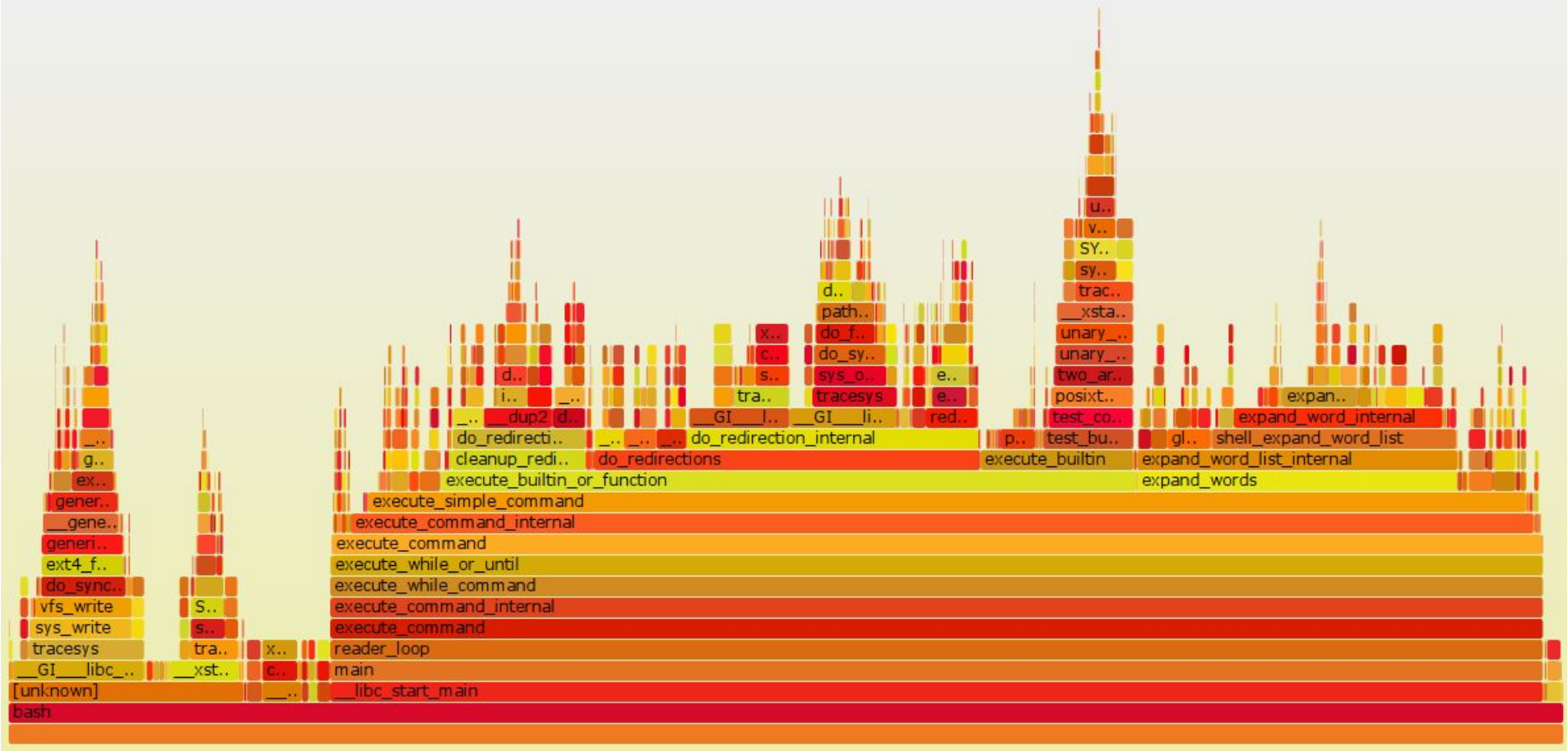
perf top

```
Samples: 3M of event 'cycles', Event count (approx.): 23
 32.80%  postgres      [.] list_nth
 20.29%  postgres      [.] ResourceOwnerForgetRelationRef
 12.87%  postgres      [.] find_all_inheritors
  7.90%  postgres      [.] get_tabstat_entry
  6.68%  postgres      [.] ResourceOwnerForgetTupleDesc
  1.17%  postgres      [.] hash_search_with_hash_value
  0.84%  postgres      [.] SearchCatCache
  0.65%  postgres      [.] AllocSetAlloc
  0.43%  [kernel]      [k] clear_page_c_e
  0.42%  libc-2.19.so  [.] lseek64
  0.36%  postgres      [.] ExecInitExpr
```

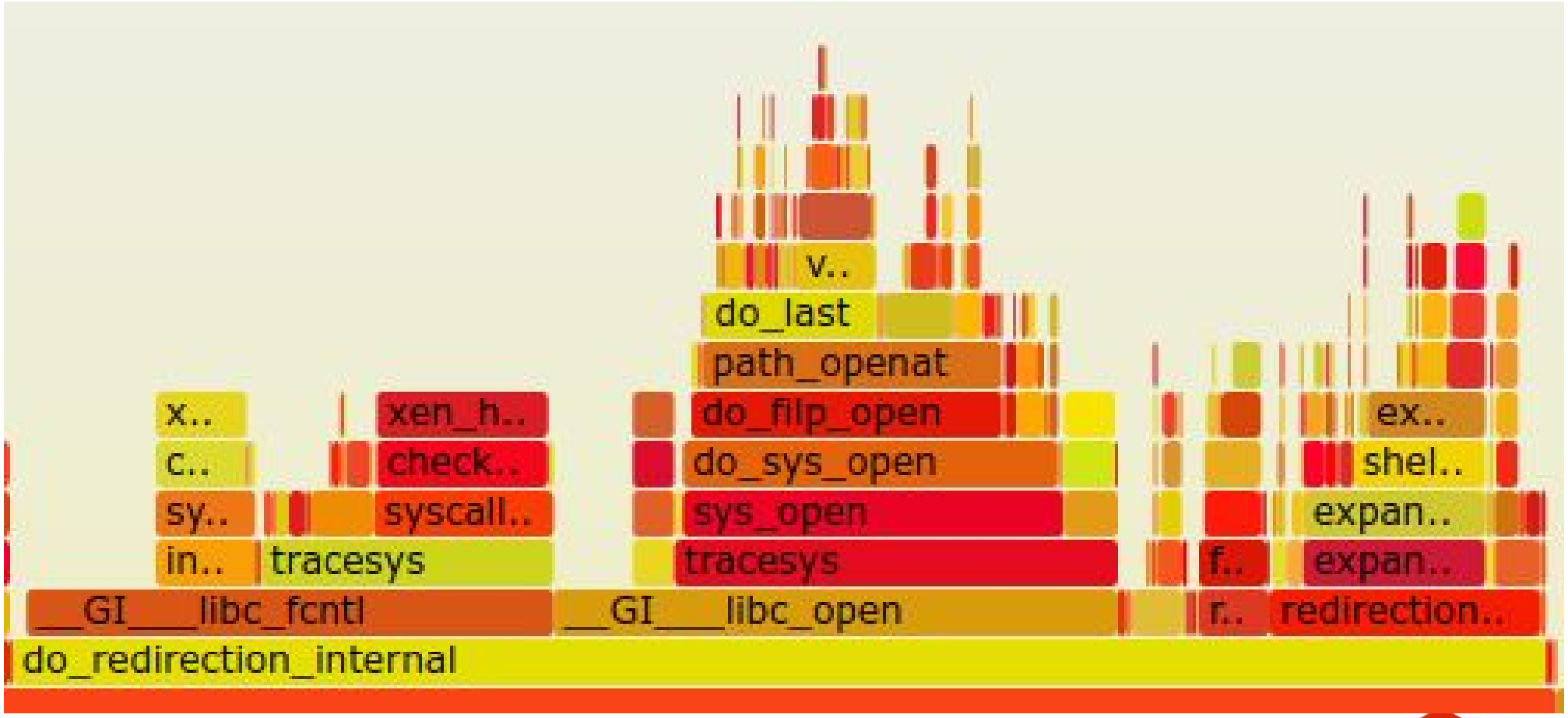
perf top — реальный патч (cc988fbb)

```
commit cc988fbb0bf60a83b628b5615e6bade5ae9ae6f4
Author: Tom Lane <tgl@sss.pgh.pa.us>
Date:   Tue Jan 26 15:20:22 2016 -0500

    Improve ResourceOwners' behavior for large numbers of owned objects.

    The original coding was quite fast so long as objects were always
    released in reverse order of addition; otherwise, it degenerated into
    O(N^2) behavior due to searching for the array element to delete.
    Improve matters by switching to hashed storage when the number of
    objects of a given type exceeds 64.  (The cutover point is open to
    discussion, of course, but some simple performance testing suggests
    that hashing has enough overhead to be a loser below there.)

    Also, refactor resowner.c so that we don't need N copies of the array
    management code.  Since all the resource IDs the code currently needs
    to deal with are either pointers or integers, it seems sufficient to
    create a one-size-fits-all infrastructure in which everything is
    converted to a Datum for storage.

    Aleksander Alekseev, reviewed by Stas Kelvich, further fixes by me
```

perf record и флеймграфы [1 / 2]



perf record и флеймграфы [2 / 2]



Brendan Gregg

http://youtu.be/tDacjrSCeq4 (Shouting in the Datacenter)

pmcstat

PMC = Performance Monitoring Counters

Тоже умеет:

- Топ самых “горячих” процедур
- Флеймграфы
- Граф вызовов как у gprof
- Только FreeBSD (под MacOS нет)

http://eax.me/freebsd-pmcstat/

SystemTap, пример скрипта

```
#!/usr/bin/env stap

probe begin
{
  printf("Start\n")
}

probe kernel.function("ip_rcv")
{
  printf("ip_rcv(%s)\n",$$parms)
}

probe timer.ms(5000)
{
  exit()
}

probe end
{
  printf("End\n")
}
```

SystemTap, плюсы/минусы

- Мощный и при этом безопасный скриптовый язык
- Код транслируется в C, компилируется в модуль ядра
- Автоматический вывод типов (строки, числа) при компиляции
- Стремно использовать в продакшне
- Больше вообще про трассировку-отладку, чем профайлинг ...
- … тем более учитывая, что есть perf

DTrace, пример скрипта

```
#!/usr/sbin/dtrace -s

syscall:freebsd:poll:entry /execname == "postgres"/
{
  printf("fds = %p, nfds = %d, timeout = %d", arg0, arg1, arg2);
}

syscall:freebsd:poll:return /execname == "postgres"/
{
  printf("result = %p", arg1)
}
```

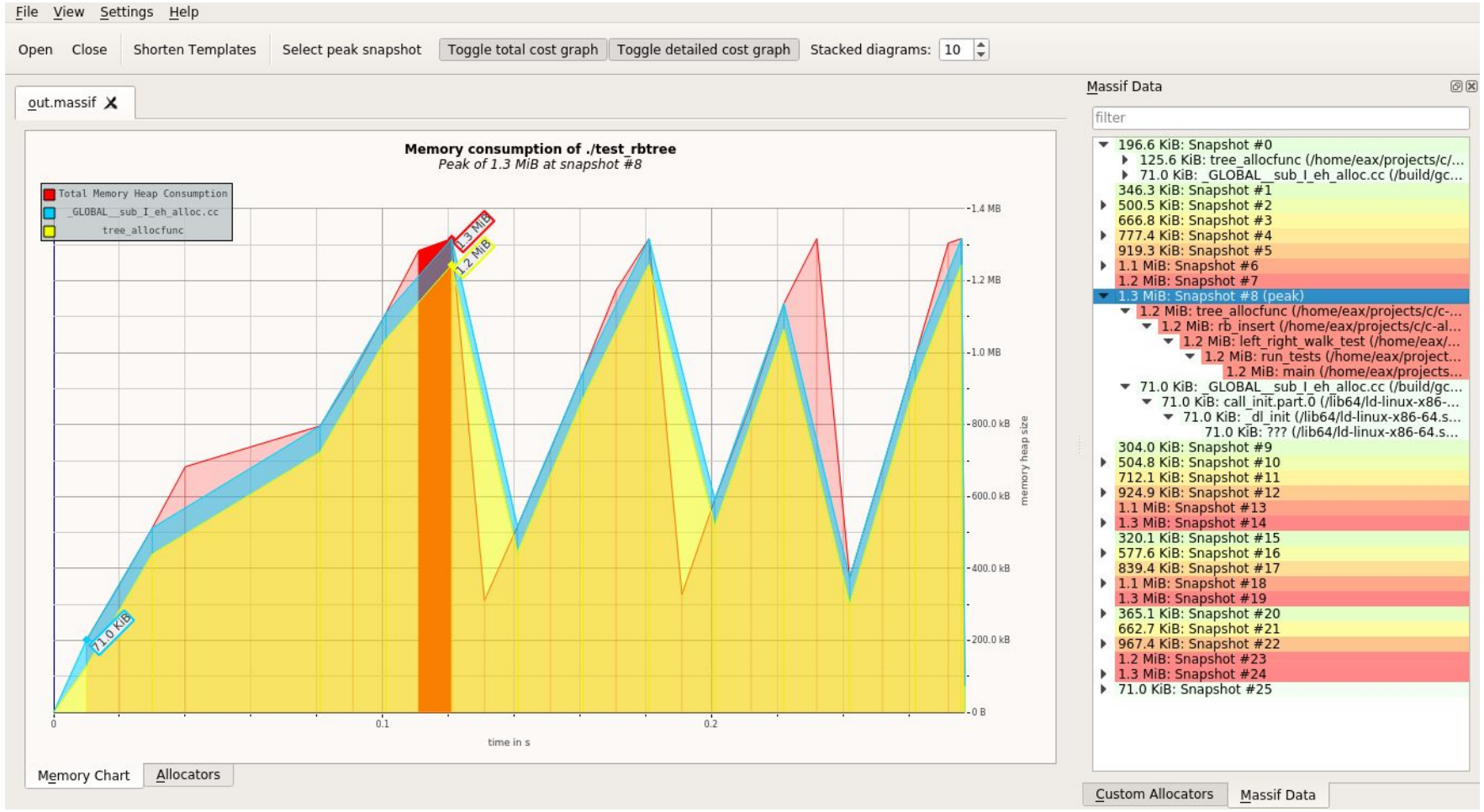
DTrace, плюсы/минусы

- Работает не только при наличии в приложении пробов
- MacOS, FreeBSD, Oracle Linux
- Не нужно компилировать и устанавливать, есть в системе из коробки
- Можно использовать на продакшне (по крайней мере, во FreeBSD)
- Есть порт для остальных дистрибутивов Linux (dtrace4linux)

HeapTrack: пример отчета

```
PEAK MEMORY CONSUMERS

4.98MB peak memory consumed over 31111 calls from
tree_allocfunc
  at /home/eax/projects/c/c-algorithms/test/struct/test_rbtree.c:37
  in /home/eax/projects/c/c-algorithms/build/test/struct/test_rbtree
1.24MB consumed over 7777 calls from:
    rb_insert
      in /home/eax/projects/c/c-algorithms/build/test/struct/test_rbtree
    left_right_walk_test
      at /home/eax/projects/c/c-algorithms/test/struct/test_rbtree.c:166
      in /home/eax/projects/c/c-algorithms/build/test/struct/test_rbtree
```

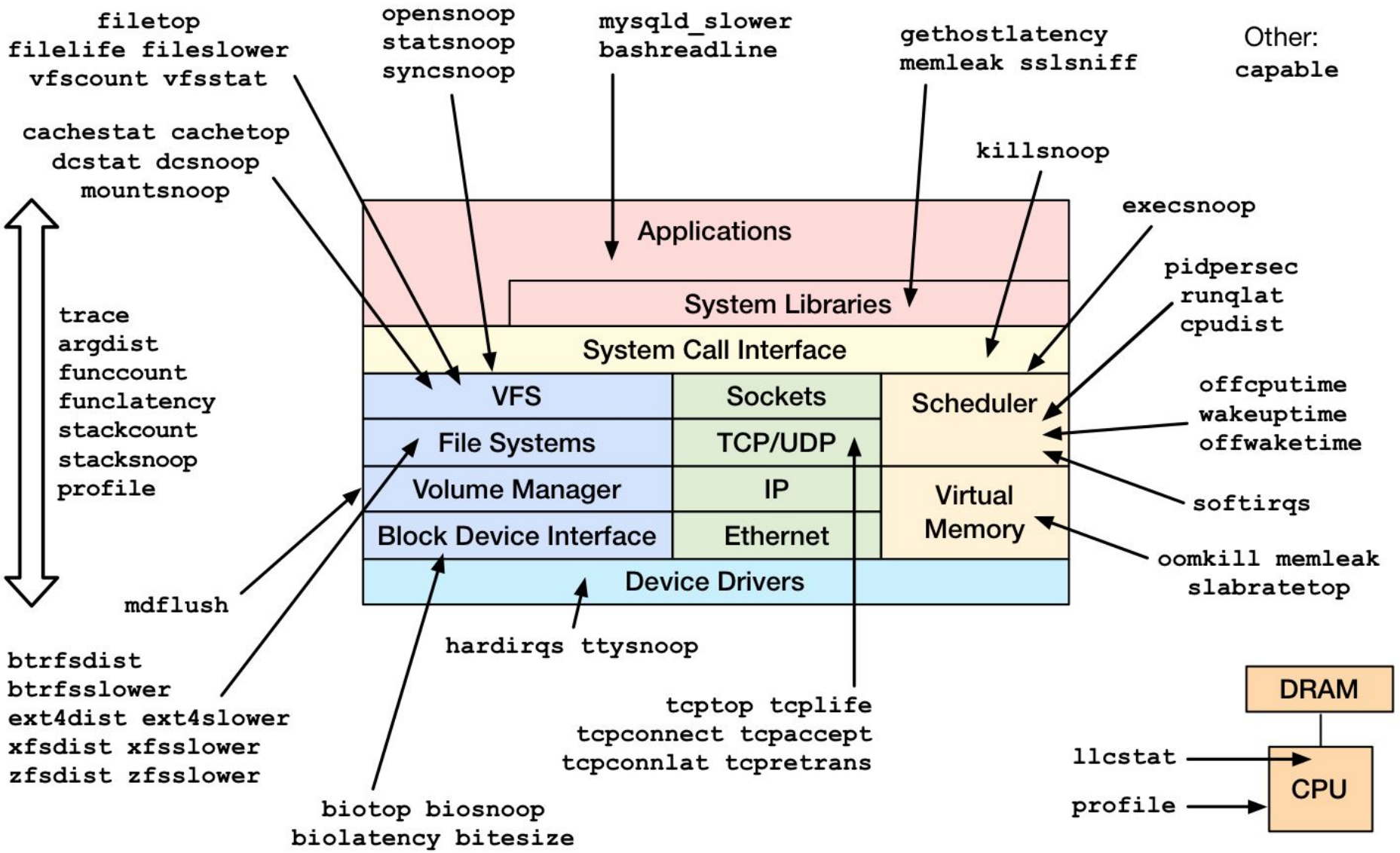
HeapTrack: Massif Visualizer



HeapTrack: плюсы/минусы

- Быстрый!
- Может цепляться к запущенным процессам
- Красивые и наглядные отчеты (есть флеймграфы)
- Умеет находить мемори лики (--print-leaks)
- Статистика по размерам выделяемой памяти (--print-histogram)
- Не знает про выделение памяти на стеке
- Только Linux (попробуйте Valgrind Massif)

Linux 4.9: BPF / bcc

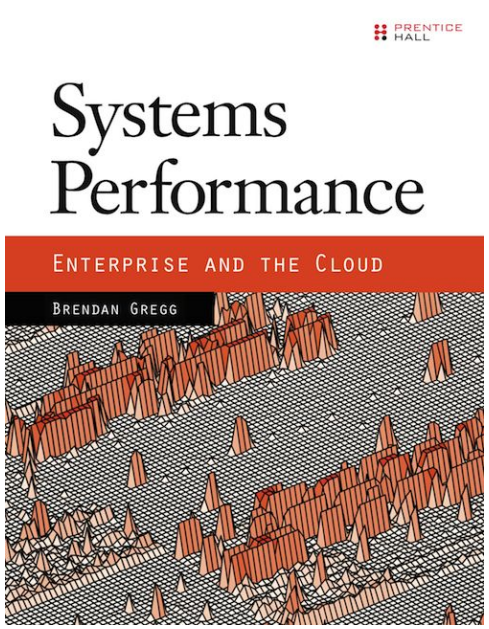


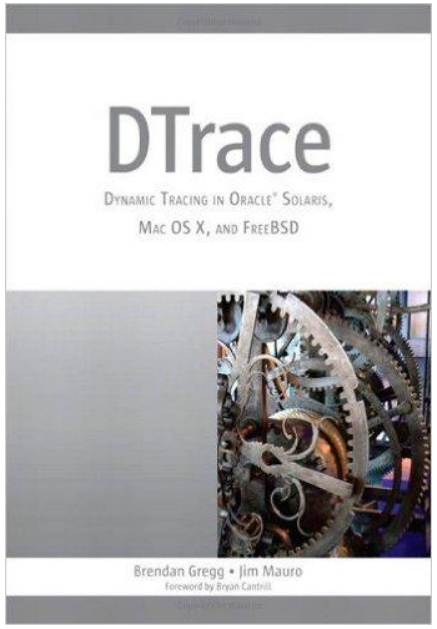
Что и когда использовать

- gettimeofday
- strace, ltrace, truss
- gprof
- **gdb / lldb** — в случае lock contention
- **perf** — если уперлись в CPU, Linux
- **pmcstat** — если уперлись в CPU, FreeBSD
- SystemTap
- **DTrace** — профайлинг на MacOS / FreeBSD + сеть, диск и т.д.
- **HeapTrack** — профилирование использования памяти на Linux
- **Valgrind Massif** — профилирование использования памяти на всем остальном
- **BPF / bcc** — профайлинг на Linux + сеть, диск и т.д.

Книги

- **Systems Performance: Enterprise and the Cloud**
  by Brendan Gregg (2013)
- **DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD**
  by Brendan Gregg, Jim Mauro (2011)





Онлайн-ресурсы

- http://www.brendangregg.com/blog/index.html
- http://dtrace.org/blogs/
- https://sourceware.org/systemtap/
- https://www.freebsd.org/doc/handbook/dtrace.html
- https://wiki.freebsd.org/DTrace
- http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Вопросы?