Программа для анализа производительности - perf

Оглавление

Счётчики Linux образуют основанную на возможностях ядра подсистему, позволяющую анализировать производительность различных операций в ядре и пользовательском пространстве. Они охватывают аппаратный уровень (CPU/PMU[1]) и программные компоненты (счётчики, точки прерывания).

Синтаксис

```
perf [--version] [--help] [OPTIONS] COMMAND [ARGS]
```

Опции

--debug

Устанавливает для отладочных переменных значения от 0 до 10, позволяя устанавливать перечисленные ниже переменные. Большее значение задаёт более подробный вывод.
verbose - базовые отладочные сообщения;
ordered-events - отладочные сообщения упорядоченных событий;
data-convert - отладочные сообщения команд преобразования данных;
stderr - запись отладочных сообщений (опция -v) на stderr в режиме браузера.

--buildid-dir

Задаёт каталог кэширования идентификаторов сборки (эта опция имеет более высокий приоритет по сравнению с переменной buildid.dir в файле конфигурации).

-v, --version

Выводит версию программы perf.

-h, --help

Выводит справку о команде perf.

Команды

bench

Общая база для тестов производительности.

Синтаксис

```
perf bench [<common options>] <subsystem> <suite> [<options>]
```

Опции общего назначения

-r, --repeat=

Задаёт число повторения тестов (по умолчанию 10).

-f, --format=

Задаёт стиль форматирования вывода.

default

Принятый по умолчанию стиль, удобный для восприятия человеком.

```
% perf bench sched pipe                       # стиль не задан
(executing 1000000 pipe operations between two tasks)
        Total time:5.855 sec
                5.855061 usecs/op
                170792 ops/sec
```

simple

Упрощённый стиль, удобный для обработки в программных сценариях.

```
% perf bench --format=simple sched pipe      # заданный стиль
5.988
```

Подсистемы

sched

Механизмы планировщиков и IPC[2].

mem

Производительность доступа к памяти.

numa

Планирование NUMA[3] и производительность MM.

futex

Стрессовые тесты Futex.

epoll

Стрессовые тесты опроса событий (epoll).

all

Все подсистемы тестирования.

Тесты для sched

messaging

Набор тестов для оценки производительности механизмов планирования и IPC, основанный на разработках Rusty Russell.

Опции messaging

-p, --pipe

Использовать pipe() вместо socketpair().

-t, --thread

Использовать множество потоков (thread) вместо множества процессов.

-g, --group=

Задаёт число групп.

[1]Performance Monitoring Unit - блок мониторинга производительности.
[2]Inter-Process Communication - коммуникации между процессами.
[3]Non-Uniform Memory Access - неоднородный доступ к памяти.

-l, --nr_loops=
Задаёт число циклов.

Пример messaging

```
% perf bench sched messaging                 # параметры по умолчанию
options (20 sender and receiver processes per group)
(10 groups == 400 processes run)

      Total time:0.308 sec

% perf bench sched messaging -t -g 20        # множество потоков, 20 групп
(20 sender and receiver threads per group)
(20 groups == 800 threads run)

      Total time:0.582 sec
```

pipe
Тесты для системных вызовов pipe() на основе pipe-test-1m.c от Ingo Molnar.

Опции pipe

-l, --loop=
Задаёт число циклов.

Пример pipe

```
% perf bench sched pipe
(executing 1000000 pipe operations between two tasks)

        Total time:8.091 sec
                8.091833 usecs/op
                123581 ops/sec

% perf bench sched pipe -l 1000              # loop 1000
(executing 1000 pipe operations between two tasks)

        Total time:0.016 sec
                16.948000 usecs/op
                59004 ops/sec
```

Тесты для mem

memcpy
Набор тестов для оценки скорости простых операций копирования в памяти различными способами.

Опции memcpy

-l, --size
Размер копируемого блока (по умолчанию 1 Мбайт). Задаётся числом с суффиксом B, KB, MB, GB или TB (без учёта регистра символов).
-f, --function
Задаёт функцию копирования (по умолчанию default). Доступные функции зависят от архитектуры, для x86-64 поддерживаются x86-64-unrolled, x86-64-movsq и x86-64-movsb.
-l, --nr_loops
Задаёт повтор вызова memcpy указанное число раз.
-c, --cycles
Задаёт использование perf события cpu-cycles вместо системного вызова gettimeofday.

memset
Набор тестов для оценки скорости простых операций установки памяти различными способами.

Опции для memset

-l, --size
Размер копируемого блока (по умолчанию 1 Мбайт). Задаётся числом с суффиксом B, KB, MB, GB или TB (без учёта регистра символов).
-f, --function
Задаёт функцию копирования (по умолчанию default). Доступные функции зависят от архитектуры, для x86-64 поддерживаются x86-64-unrolled, x86-64-stosq и x86-64-stosb.
-l, --nr_loops
Задаёт повтор вызова memset указанное число раз.
-c, --cycles
Задаёт использование perf события cpu-cycles вместо системного вызова gettimeofday.

Тесты для numa

mem
Набор тестов для оценки выполнения заданий NUMA.

Тесты для futex

hash
Тесты для оценки хэш-таблиц.
wake
Тесты для оценки вызовов wake.
wake-parallel
Тесты для оценки параллельных вызовов wake.
requeue
Тесты для оценки вызовов requeue.
lock-pi
Тесты для оценки вызовов futex lock_pi.

Опции для всех тестов

-S, --shared
Использовать вызовы общих futex вместо приватных.

-s, --silent
Без вывода данных и деталей.

-t, --threads <n>
Задаёт число потоков (thread)

Опции для отдельных тестов

-f, --futexes <n>
Задаёт число вызовов futex на поток (только hash).

-r, --runtime <n>
Задаёт время работы в секундах (hash и lock-pi).

-w, --nwakes <n>
Задаёт число потоков для разового пробуждения (wake и wake-parallel)

-q, --nrequeue <n>
Задаёт число потоков для разового изменения очерёдности (только requeue).

-M, --multi
Использовать множество futex (только lock-pi).

Тесты для epoll

wait
Тесты для оценки одновременных вызовов epoll_wait.

ctl
Тесты для оценки множества вызовов epoll_ctl.

Опции для всех тестов

-f, --nfds <n>
Задаёт число файловых дескрипторов для отслеживания каждым потоком.

-N, --nested <n>
Задаёт уровень вложенности для иерархии epoll (по умолчанию без вложенности - 0).

-n, --noaffinity
Отключает близость CPU (affinity).

-R, --randomize
Задаёт выполнение случайных операций на случайных дескрипторах файлов.

-r, --runtime <n>
Задаёт время работы в секундах.

-t, --threads <n>
Задаёт число потоков (thread).

-v, --verbose
Расширенный вывод

Опции для wait

-B, --nonblocking
Неблокируемое поведение epoll_wait.

-E, --edge
использовать интерфейс Edge-triggered (по умолчанию LT).

-m, --multiq
Использовать множество экземпляров epoll (по одному на поток).

-S, --oneshot
Использовать семантику EPOLLONESHOT

stat

Выполняет указанную параметром команду, собирая статистику работы.

Синтаксис

```
perf stat [-e <EVENT> | --event=EVENT] [-a] <command>
perf stat [-e <EVENT> | --event=EVENT] [-a] - <command> [<options>]
perf stat [-e <EVENT> | --event=EVENT] [-a] record [-o file] - <command> [<options>]
perf stat report [-i file]
```

Опции

<command>…
Любая команда, которая может быть выполнена в консоли.

record
См. Запись статистики (record)

report
См. Статистический отчёт (report)

-e, --event=
Задаёт событие PMU, позволяя выбирать указанные ниже варианты.

- Символическое имя события (perf list позволяет увидеть список всех событий).
- Необработанное событие PMU (eventsel+umask) в форме rNNN, где NNN является шестнадцатеричным дескриптором события.
- Символически сформированное имя вида pmu/param1=0x3,param2/, где param1 и param2 определены как форматы для PMU в файлах /sys/bus/event_source/devices/<pmu>/format/*. Классификатор percore задаёт суммирование счётчиков для обоих аппаратных потоков (thread) в ядре. Например,
  `perf stat -A -a -e cpu/event,percore=1/,otherevent ...`
- Символически сформированное имя вида pmu/config=M,config1=N,config2=K/, где M, N, K - числа в десятичном, шестнадцатеричном или восьмеричном формате. Приемлемые значения параметров config, config1 и config2 определяются записями в файлах /sys/bus/event_source/devices/<pmu>/format/*

Отметим, что два последних варианта поддерживают префиксы и шаблоны (glob) в именах PMU для упрощённого указания событий во множестве экземпляров одного PMU в больших системах (например, PMU контроллера памяти). Множественные экземпляры типичны для неосновных PMU, поэтому при сопоставлении префикс uncore_ не принимается во внимание.

-i, --no-inherit

Дочерние задачи не наследуют счётчики.

-p, --pid=<pid>

События для процесса с заданными идентификаторами (список разделённых запятыми значений).

-t, --tid=<tid>

События для потоков (thread) с заданными идентификаторами (список разделённых запятыми значений).

-a, --all-cpus

Сбор в масштабе системы от всех CPU (используется по умолчанию, если цель не задана).

--no-scale

Отключает масштабирование и нормализацию значений счётчиков.

-d, --detailed

Задаёт уровень детализации вывода:
-d - подробные события, кэш данных L1 и LLC;
-d -d - более подробные события, dTLB и iTLB;
-d -d -d - максимально подробные события, добавление событий предварительной выборки.

-r, --repeat=<n>

Повтор команды заданное число раз (не более 100) с усреднением и указанием стандартного отклонения. 0 означает бесконечный повтор.

-B, --big-num

Задаёт вывод больших чисел с отделением тысяч в соответствии с установками locale.

-C, --cpu=

Задаёт учёт лишь указанных CPU, которые можно задать номерами через запятые без пробелов (0,1) или как диапазон (0-2). В режиме учёта по потокам эта опция игнорируется. Для мониторинга в масштабе системы сохраняется актуальность опции -a. По умолчанию учитываются все процессоры.

-A, --no-aggr

Отключает агрегирование счётчиков для отслеживаемых CPU.

-n, --null

Пустой прогон без запуска счётчиков.

-v, --verbose

Задаёт подробный вывод (показывает ошибки открытия счётчиков и т. п.).

-x SEP, --field-separator SEP

Задаёт вывод счётчиков с использованием стиля CSV для упрощения импорта в электронные таблицы. Колонки разделяются строкой, заданной параметром SEP.

--table

Задаёт вывод времени для каждого запуска (опция -r) в форме таблицы. Например, по команде

```
$ perf stat --null -r 5 --table perf bench sched pipe
```

Будут выведены счётчики статистики для 5 запусков perf bench sched pipe и общий результат

```
# Table of individual measurements:
5.189 (-0.293) #
5.189 (-0.294) #
5.186 (-0.296) #
5.663 (+0.181) ##
6.186 (+0.703) ####

# Final result:
5.483 +- 0.198 seconds time elapsed  ( +-  3.62% )
```

-G name, --cgroup name

Задаёт отслеживание лишь контейнера (cgroup) с именем name. Эта опция доступна лишь в режиме отслеживания по процессорам (per-cpu). Файловая система cgroup должна быть примонтирована. Все потоки, относящиеся в контейнеру name, будут отслеживаться при работе на контролируемых CPU. Можно указать несколько cgroup, каждая из которых будет применяться к соответствующему событию, т. е. первая cgroup будет применяться к первому событию, вторая - ко второму и т. д. Можно представить пустую cgroup (отслеживается все время), используя, например, -G foo,,bar. Группы cgroup должны иметь соответствующие события, т. е. они всегда указывают на события, определённые ранее в командой строке. Если нужно отслеживать множество событий для определённой cgroup, можно использовать -e e1 -e e2 -G foo,foo или просто -e e1 -e e2 -G foo.

Если нужно отслеживать, например, cycles для cgroup, а также для всей системы, можно воспользоваться командой perf stat -e cycles -G cgroup_name -a -e cycles.

-o file, --output file

Задаёт вывод данных в указанный файл.

--append

Задаёт добавление вывода в конец файла, заданного опцией -o (без этой опции игнорируется).

--log-fd

Задаёт вывод в файл с дескриптором fd вместо stderr и является взаимоисключающей с опцией -output. Позволяет использовать также опцию --append.

--pre, --post

Задаёт «ловушки» перед измерением и после него.

-I msecs, --interval-print msecs

Задаёт вывод приращения счётчиков каждые N мсек (минимум 1). В некоторых случаях издержки могут быть велики (например, при интервалах меньше 100 мсек). Опцию следует применять с осторожностью.

--interval-count times

Задаёт вывод приращения счётчиков фиксированное число раз. Эту опцию следует применять вместе с -I. Например, perf stat -I 1000 --interval-count 2 -e cycles -a.

--interval-clear

Задаёт очистку экрана перед следующим интервалом.

--timeout msecs
Останавливает сеанс статистики perf и выводит приращения счётчиков по истечении N мсек (минимум 10). Не поддерживается с опцией -I.

--metric-only
Задаёт вывод только расчётных параметров (в одну строку). Не поддерживается с опцией --per-thread.

--per-socket
Объединяет счётчики отдельных физических процессоров для измерений в масштабе системы. Это полезно для обнаружения разбалансировки между процессорами. Для включения режима опция --per-socket дополняется опцией -a. Вывод включает номер процессора и число активных ядер в нем.

--per-core
Объединяет счётчики ядер отдельных физических процессоров. Это полезно для обнаружения разбалансировки между ядрами. Для включения режима опция --per-core дополняется опцией -a. Вывод включает номер ядра и число активных логических процессоров на данном физическом процессоре.

--per-thread
Объединяет счётчики отслеживаемых потоков (thread) при мониторинге потоков (-t) или процессоров (-p).

-D msecs, --delay msecs
Задаёт ожидание перед началом измерений после запуска программы. Это полезно для исключения стартовой фазы программы, которая зачастую отличается от основного процесса.

-T, --transaction
Выводит статистику выполнения транзакций, если они поддерживаются.

Запись статистики (record)

Сохраняет статистические данные в указанном файле.

-o file, --output file
задаёт имя выходного файла.

Статистический отчёт (report)

Считывает данные и выводит отчёт на основании файла данных perf.

-i file, --input file
Имя входного файла данных.

--per-socket
Объединение счётчиков по физическим процессорам для измерений в масштабе системы.

--per-core
Объединение счётчиков по ядрам одного физического процессора для измерений в масштабе системы.

-M, --metrics[1]
Задаёт вывод параметров или групп параметров, указанных в разделённом запятыми списке. Для группы добавляются все параметры данной группы. События для параметров учитываются автоматически. Возможные параметры и группы можно увидеть с помощью команды perf list.

-A, --no-aggr
Отключает объединение данных для всех отслеживаемых CPU.

--topdown[1]
Выводит параметры метрики уровня 1, сообщая об узких местах при обработке, если это поддерживается процессором. Это позволяет найти узкие места в конвейере CPU для привязанных к CPU заданий путём разрыва циклов, связанных с пользовательскими (frontend) или аппаратными (backend) ограничениями, неверными предсказаниями (bad speculation) и сокрытием (retiring).
Пользовательское ограничение (Frontend bound) означает, что CPU не может извлекать и декодировать инструкции достаточно быстро. Аппаратное ограничение (Backend bound) говорит об узких местах в расчётах или доступе к памяти. Неверные предсказания (Bad Speculation) указывают на то, что CPU использует ненужные циклы в результате ошибочного предсказания ветвлений и т. п. Retiring означает, что CPU работает без явных узких мест. Узкое место является реальной пробкой лишь в том случае, когда задание действительно ограничено CPU, а не чем-то иным.
Для получения лучших результатов обычно хорошо применять интервал -I 1000, поскольку узкие места в заданиях могут меняться достаточно часто.
Параметры узких мест собираются на уровне ядер, а не потоков CPU. Режим работы на уровне ядер включается автоматически и нужна опция -a (глобальный мониторинг), что требует полномочий root или установки perf.perf_event_paranoid=-1.
Режим topdown полностью использует блок PMU и нужно отключить NMI watchdog (от имени root) командой echo 0 > /proc/sys/kernel/nmi_watchdog для получения лучших результатов. В противном случае могут возникать пробки, не совместимые с рабочей нагрузкой.
Эта опция включает режим --metric-only, если нет опции --no-metric-only.
Для интерпретации результатов обычно нужно знать, на каких процессорах выполняется работа. При необходимости процессоры можно задать принудительно с помощью taskset.

--no-merge[2]
Отменяет слияние результатов от одного PMU.
При создании множества событий по одной спецификации статистика будет по умолчанию объединять счётчики событий и выводить результат в одну строку. Данная опция отменяет это и обеспечивает вывод отдельных событий и счётчиков.
Множество событий создаётся по одной спецификации, если (1) префикс или регулярное выражение glob применяется для имени PMU или (2) используются псевдонимы, указанные после событий Kernel PMU.

--smi-cost[2]
Измерение стоимости SMI[3], если поддерживаются события msr/aperf/ и msr/smi/. В процессе измерения устанавливается опция в файле /sys/device/cpu/freeze_on_smi для замораживания счётчиков при SMI. Эта установке не влияет на счётчик aperf и стоимость SMI может быть измерена (aperf - непрерванные циклы ядра).

[1]Эта опция в реальности не поддерживается для режима report.
[2]Эта опция в реальности не поддерживается для режима report.
[3]System Management Interrupt - прерывание для системных измерений

На практике процент циклов SMI очень полезен для ориентированного на производительность анализа. По умолчанию применяется опция --metric_only. Выходом является % циклов SMI, равный (aperf - непрерванные циклы ядра) / aperf.

При желании получить актуальное значение можно воспользоваться опцией --no-metric-only.

Пример

```
$ perf stat make
[...]
Performance counter stats for 'make':

          83723.452481      task-clock:u (msec)       #    1.004 CPUs utilized
                     0      context-switches:u        #    0.000 K/sec
                     0      cpu-migrations:u          #    0.000 K/sec
             3,228,188      page-faults:u             #    0.039 M/sec
       229,570,665,834      cycles:u                  #    2.742 GHz
       313,163,853,778      instructions:u            #    1.36  insn per cycle
        69,704,684,856      branches:u                #  832.559 M/sec
         2,078,861,393      branch-misses:u           #    2.98% of all branches

       83.409183620 seconds time elapsed

       74.684747000 seconds user
        8.739217000 seconds sys
```

В этом примере видны три типа временных показателей. Всегда выводится время, в течение которого счётчики были включены (активны)

```
       83.409183620 seconds time elapsed
```

Для сеансов рабочей нагрузки указывается также время, затраченное в пользовательском и системном пространстве

```
       74.684747000 seconds user
        8.739217000 seconds sys
```

Эти значения совпадают с выводимыми утилитой time.

Формат CSV

С опцией -x команда perf stat будет выводить результаты в формате CSV. Запятые при выводе не будут помещаться в кавычки. Для упрощения анализа вывода рекомендуется использовать с опцией -x другие разделители (например, \;).

Поля выводятся в следующем порядке:

- необязательная временная метка (в мксек) в дробной части секунд (с -I xxx);
- необязательный идентификатор CPU, ядра или сокета;
- необязательное число агрегированных логических CPU;
- значение счётчика;
- единица измерения счётчика или пустое значение;
- имя события;
- время работы счётчика;
- доля времени измерения от времени работы счётчика (%);
- необязательное отклонение при сборе множества значений с опцией -r;
- необязательное значение параметра;
- необязательная единица измерения параметра.

После этих полей могут выводиться дополнительные параметры.

top

Создание и вывод профиля производительности в реальном масштабе времени.

Синтаксис

```
perf top [-e <EVENT> | --event=EVENT] [<options>]
```

Опции

-a, --all-cpus

Сбор данных на уровне системы в целом (используется по умолчанию)

-c <count>, --count=<count>

Период выборки событий.

-C <cpu-list>, --cpu=<cpu>

Отслеживание только указанных CPU. Процессоры можно задать списком разделённых запятыми идентификаторов (без пробелов, 0,1) или диапазоном (0-2). По умолчанию отслеживаются все CPU.

-d <seconds>, --delay=<seconds>

Интервал обновления в секундах.

-e <event>, --event=<event>

Задаёт событие PMU, которое может быть указано символьным именем (см. perf list) или необработанным (raw) событием PMU (eventsel+umask) в форме rNNN, где NNN - шестнадцатеричный дескриптор события.

-E <entries>, --entries=<entries>

Задаёт вывод указанного числа функций.

-f <count>, --count-filter=<count>

Задаёт вывод лишь функций, где число событий превосходит заданное значение.

--group
Задаёт группировку счётчиков.

-F <freq>, --freq=<freq>
Задаёт частоту профилирования. Для профилирования с максимально возможной частотой используется значение max, т. е. частота, указанная в kernel.perf_event_max_sample_rate (sysctl).

-i, --inherit
Дочерние задачи не наследуют счётчики.

-k <path>, --vmlinux=<path>
Путь к файлу vmlinux, требуемый для аннотирования.

--ignore-vmlinux
Задаёт игнорирование файлов vmlinux.

--kallsyms=<file>
Указывает путь к файлу kallsyms.

-m <pages>, --mmap-pages=<pages>
Число страниц данных mmap (должно быть степенью 2) или размер с суффиксом, указывающим единицы (B/K/M/G). Размер округляется до ближайшего значения, равного степени 2.

-p <pid>, --pid=<pid>
События профиля для имеющихся идентификаторов процессов (разделённые запятыми значения).

-t <tid>, --tid=<tid>
События профиля для имеющихся идентификаторов потоков (разделённые запятыми значения).

-u, --uid=
Запись событий в потоках, принадлежащих uid (имя или значение).

-r <priority>, --realtime=<priority>
Сбор данных с указанным приоритетом RT SCHED_FIFO.

--sym-annotate=<symbol>
Аннотирование этого символа.

-K, --hide_kernel_symbols
Сокрытие символов ядра.

-U, --hide_user_symbols
Сокрытие пользовательских символов.

--demangle-kernel
Разбирать символы ядра.

-D, --dump-symtab
Дамп таблицы символов, используемой для профилирования.

-v, --verbose
Задаёт подробный вывод (ошибки открытия счётчиков и т. п.).

-z, --zero
Задаёт сброс истории при обновлении экрана.

-s, --sort
Задаёт сортировку по ключу - pid, comm, dso[1], symbol, parent, srcline, weight, local_weight, abort, in_tx, transaction, overhead, sample, period (см. report).

--fields=
Задаёт поля вывода. Множество полей можно указать в формате CSV. Доступны поля overhead, overhead_sys, overhead_us, overhead_children, sample и period. Можно указать также ключи сортировки. По умолчанию, ключи сортировки, не заданные в поле --fields, будут добавляться автоматически.

-n, --show-nr-samples
Задаёт вывод столбца с числом выборок.

--show-total-period
Задаёт вывод столбца с суммой периодов.

--dsos
Задаёт рассмотрение лишь символов указанных объектов dso, влияя на % в столбце overhead (см. --percentage).

--comms
Задаёт рассмотрение лишь символов в указанных столбцах, влияя на % в столбце overhead (см. --percentage).

--symbols
Задаёт рассмотрение лишь указанных символов, влияя на % в столбце overhead (см. --percentage).

-M, --disassembler-style=
Задаёт стиль дизассемблирования для objdump.

--source
Задаёт чередование исходного кода с ассемблерным. По умолчанию включена и отключается опцией --no-source.

--asm-raw
Показывает необработанное кодирование ассемблерных инструкций.

-g
Включает запись графа вызовов (stack chain/backtrace).

--call-graph [mode,type,min[,limit],order[,key][,branch]]
Устанавливает и включает запись графа вызовов, подразумевая опцию -g (см. --call-graph в record и report).

--children
Накапливает цепочку вызовов дочернего процесса к родительской записи, чтобы их можно было сопоставить в выводе. Вывод будет включать столбец Children и сортироваться по его значению (нужна опция -g или --call-graph). Подробности приведены в разделе Расчёт издержек. По умолчанию опция включена и отключается --no-children.

--max-stack
Устанавливает предел глубины стека при разборе цепочки вызовов, устанавливая компромисс между потерей деталей и скоростью обработки, особенно для задач с большой глубиной стека вызовов. По умолчанию используется значение 127 или содержимое файла /proc/sys/kernel/perf_event_max_stack, если он имеется.

--ignore-callees=<regex>
Задаёт игнорирование вызывающих функции, указанные регулярным выражением regex. Это оказывает влияние на сбор статистики о вызывающих такие функции, указывая их лишь один раз в дереве графа вызовов.

[1]Dynamic shared object - совместно используемый динамический объект.

--percent-limit

Отключает вывод записей с издержками ниже указанных (по умолчанию 0).

--percentage relative|absolute

Определяет вывод процента издержек отфильтрованных записей. Фильтры могут применяться с помощью опций --comms, --dsos и,или --symbols, а также операций Zoom в TUI[1] (thread, dso и т. п.).

Относительный вариант (relative) означает, что издержки рассчитываются только для отфильтрованных записей, поэтому сумма показанных значений будет составлять 100%. При абсолютном варианте (absolute) применяется исходное значение до и после применения фильтра.

-w, --column-widths=<width[,width...]>

Задаёт ширину каждой колонки для удобочитаемости. 0 отменяет ограничение и используется по умолчанию.

--proc-map-timeout

Обработка уже имеющихся потоков /proc/XXX/mmap может занять продолжительное время и для таких случаев нужно установить время ожидания. По умолчанию тайм-аут составляет 500 мсек.

-b, --branch-any

Включает выборку из стека ветвления. Выбираться могут любые типы ветвей. Эта опция является сокращением для --branch-filter any (см. --branch-filter).

-j, --branch-filter

Включает выборку из стека ветвления. Каждая выборка включает набор последовательных ветвлений. Число фиксируемых ветвлений зависит от базового оборудования, типа интересующих ветвлений и исполняемого кода. Можно выбрать типы ветвлений с помощью фильтров. Полный список модификаторов приведён в описании команды record.

Опция требует указания хотя бы одного из типов any, any_call, any_ret, ind_call, cond. Уровни привилегий можно опускать и в этом случае к фильтру ветвления будет применяться уровень привилегий связанного события. Уровни привилегий как ядра (k), так и гипервизора (hv) зависят от разрешений. При выборке множества событий выборка стека ветвления включается для всех отбираемых событий. Тип отбираемого ветвления одинаков для всех событий. Разные фильтры должны указываться в виде разделённого запятыми списка --branch-filter any_ret,u,k. Отметим, что на некоторых процессорах эта опция не поддерживается.

--raw-trace

Отключает использование форматирования вывода или подключаемых модулей при выводе событий трассировки.

--hierarchy

Включает иерархический вывод.

--overwrite

Включает использование самых свежих записей. Это помогает в машинах с большим числом ядер (таких как Knights Landing/Mill), но по умолчанию отключено, поскольку используемая этим методом пауза ведёт к потере событий метаданных (таких как PERF_RECORD_MMAP), что мешает perf top преобразовать выборки и ведёт к появлению множества нераспознанных выборок в пользовательском интерфейсе. Эту опции следует включать на упомянутых машинах при профилировании работы, не создающей краткосрочных потоков и/или не использующих большого числа выполняемых операций mmap. Планируется работа по решению проблемы с паузой, но пока опция отключена по умолчанию.

--force

Отключает проверку владения.

--num-thread-synthesize

Задаёт число потоков, запускаемых при синтезе событий для имеющихся процессов. По умолчанию число потоков совпадает с числом активных CPU.

Клавиши интерактивного управления

[d]

Выводит интервал обновления.

[e]

Число выводимых записей.

[E]

Отображаемое событие при активности нескольких счётчиков.

[f]

Фильтр отображения профиля (>= счётчика обращений).

[F]

Фильтр вывода аннотаций (>= % от общего).

[s]

Аннотировать символы.

[S]

Прекратить аннотирование с возвратом вывода полного профиля.

[K]

Скрыть символы ядра.

[U]

Скрыть пользовательские символы.

[z]

Переключить режим обновления счётчиков при обновлении.

[qQ]

Завершение работы.

Нажатие любой другой клавиши приведёт к выводу меню для выбора варианта.

Расчёт издержек

Издержки (overhead) могут выводиться в столбцах Children и Self, когда perf собирает цепочки вызовов. Собственные (self) издержки рассчитываются просто путём сложения всех значений периодов для записи - обычно это функция (символ). Это будет значение, которое традиционно показывает perf и сумма всех собственных издержек будет составлять 100%.

[1]Text User Interface - текстовый пользовательский интерфейс.

Дочерние издержки рассчитываются путём сложения всех значений периодов дочерних функций, чтобы они могли отражать общие издержки функций верхнего уровня, даже они не выполняются напрямую. Дочерними считаются функции, вызываемые из других (родительских) функций.

Может показаться странным, что сумма всех дочерних издержек превышает 100%, но в них учитываются издержки потомков следующих уровней. Однако это позволяет увидеть функцию с наибольшими издержками, даже если выборки распределены по её потомкам.

Рассмотрим пример с тремя функциями.

```
void foo(void) {
    /* do something */
}

void bar(void) {
    /* do something */
    foo();
}

int main(void) {
    bar()
    return 0;
}
```

В этом случае foo является потомком bar, а bar - прямым потомком main, поэтому foo является также потомком main. Иными словами, main является предком foo и bar, а bar - предком foo.

Предположим, что все выборки выполняются только в foo и bar. При записи с помощью цепочек вызовов вывод perf будет иметь вид

```
Overhead  Symbol
........  ......................
  60.00%  foo
          |
          --- foo
              bar
              main
              __libc_start_main

  40.00%  bar
          |
          --- bar
              main
              __libc_start_main
```

При включённой опции --children значения собственных издержек дочерних функций (foo и bar) добавляются к родительским издержкам и вывод будет иметь вид

```
Children      Self  Symbol
........  ........  ....................
 100.00%     0.00%  __libc_start_main
          |
          --- __libc_start_main

 100.00%     0.00%  main
          |
          --- main
              __libc_start_main

 100.00%    40.00%  bar
          |
          --- bar
              main
              __libc_start_main

  60.00%    60.00%  foo
          |
          --- foo
              bar
              main
              __libc_start_main
```

Видно, что собственные издержки foo (60%) добавлены к дочерним издержкам bar, main и __libc_start_main. Аналогично, собственные задержки bar (40%) добавлены к дочерним издержкам main и __libc_start_main.

Таким образом, __libc_start_main и main показаны первыми, поскольку они имеют одинаковые (100%) издержки потомков (даже при нулевых собственных издержках) и являются родителями foo и bar.

Начиная с версии 3.16, дочерние издержки выводятся по умолчанию и применяются для сортировки вывода. Отключить издержки потомков можно с помощью опции --no-children в командной строке, а также добавлением report.children = false или top.children = false в конфигурационный файл perf.

record

Выполняет пользовательскую команду и записывает профиль работы в файл perf.data. Полученный файл можно затем использовать для подготовки отчёта с помощью perf report.

Синтаксис

```
perf record [-e <EVENT> | --event=EVENT] [-a] <command>
perf record [-e <EVENT> | --event=EVENT] [-a] - <command> [<options>]
```

Опции

<command>...

Любая команда, которую можно выполнить в командном процессоре.

-e, --event=

Задаёт событие PMU, которое может быть указано перечисленными ниже способами.

- Символическое имя события (perf list позволяет увидеть список всех событий).
- Необработанное событие PMU (eventsel+umask) в форме rNNN, где NNN является шестнадцатеричным дескриптором события.
- Символически сформированное имя вида pmu/param1=0x3,param2/, где param1 и param2 определены как форматы для PMU в файлах /sys/bus/event_source/devices/<pmu>/format/*.

- Символически сформированное имя вида pmu/config=M,config1=N,config2=K/, где M, N, K - числа в десятичном, шестнадцатеричном или восьмеричном формате. Приемлемые значения параметров config, config1 и config2 определяются записями в файлах /sys/bus/event_source/devices/<pmu>/format/*,
- Имеются также параметры, не определённые в .../<pmu>/format/*, которые могут применяться взамен принятых по умолчанию значений для каждого событий. Ниже перечислены некоторые из основных параметров.
  - period - задаёт период выборки для события.
  - freq - задаёт частоту выборки для события.
  - time - включает (1) и отключает (0) выборку времени (по умолчанию включена).
  - call-graph - включает и отключает граф вызовов. Приемлемы строки fp для режима FP, dwarf для режима DWARF[1], lbr для режима LBR[2] и no для отключения графа вызовов.
  - stack-size задаёт размер пользовательского стека для режима DWARF.
  - name - задаёт определённое пользователем имя события. Можно использовать одинарные кавычки (') для корректного разбора строки командным процессором. Например, name=\'CPU_CLK_UNHALTED.THREAD:cmask=0x1\'.

  Дополнительная информация представлена в описании команды perf list.
  Если пользователь явно задаёт опции, конфликтующие с параметрами, установленные параметрами значения будут переопределены.
  В файлах .../<pmu>/format/* не определены также параметры конфигурации конкретных драйверов PMU. Параметры с префиксом @ не интерпретируются в пользовательском пространстве и напрямую передаются драйверу PMU. Например,
  ```
  perf record -e some_event/@cfg1,@cfg2=config/ ...
  ```
  будет смотреть cfg1 и cfg2=config, переданные драйверу PMU, связанному с событием, для дальнейшей обработки. На параметры конфигурации ограничений не налагается, поскольку они семантически понятны и поддерживаются драйвером PMU.
- События аппаратных точек прерывания в форме \mem:addr[/len][:access], где addr указывает адрес в памяти, где требуется остановка. Параметр access указывает тип доступа к памяти (чтение, запись, выполнение) и может передаваться в форме \mem:addr[:[r][w][x]]. Параметр len указывает диапазон - число байтов, начиная с addr, которые будет охватывать точка прерывания. Если вы хотите профилировать чтение и запись в 0x1000, просто укажите mem:0x1000:rw. Для профилирования записи в [0x1000~1008) следует указать mem:0x1000/8:w.
- Исходный файл BPF[3] (например, на языке c) или скомпилированный объектный файл (.o) для выбора одного или множества событий BPF. Программа BPF может привязываться к разным событиям perf по именам разделов ELF.
  При обработке файла .c программа perf будет искать установленную LLVM для компиляции объектного файла. Можно передать необязательные опции clang с помощью опции командной строки --clang-opt. Например,
  ```
  perf record --clang-opt "-DLINUX_VERSION_CODE=0x50000" -e tests/bpf-script-example.c
  ```
  Отметим, что опция --clang-opt должна размещаться до опции --event/-e.
- Группа событий, помещённых в фигурные скобки "{event1,event2,...}". События разделяются запятыми, а группу следует помещать в кавычки для предотвращения разбора командным процессором. Может потребоваться использование опции --group в команде perf report для совместного просмотра групповых событий.

--filter=<filter>

Фильтр событий. Эту опцию следует указывать после селектора событий (-e), который указывает событие точки трассировки или аппаратную трассировку PMU (например, Intel PT или CoreSight).

- Фильтры точек трассировки.
  В этом случае опции --filter объединяются с помощью символов &&.
- Фильтры адресов
  PMU аппаратной трассировки анонсирует свою возможность доступа к фильтрам адресов путём указания отличного от 0 значения в файле /sys/bus/event_source/devices/<pmu>/nr_addr_filters.
  Фильтры адресов имеют формат
  ```
  filter|start|stop|tracestop <start> [/ <size>] [@<file name>]
  ```
  - filter - указывает область трассировки;
  - start - указывает адрес, где начинается трассировка;
  - stop - указывает адрес, где останавливается трассировка;
  - tracestop - указывает область, где останавливается трассировка.

  Параметр <file name> указывает имя объектного файла, <start> - смещение в коде для трассировки, <size> - размер трассируемой области. Фильтры start и stop не обязаны задавать <size>.
  Если объектный файл не задан, предполагается ядро и в этом случае стартовым адресом должен быть текущий адрес ядра в памяти.
  Параметр <start> можно также указать именем символа. Если это имя не уникально, можно снять неоднозначность указанием #n, где n является порядковым номером символа в упорядоченном по адресам списке. Параметры #0, #g или #G позволяют выбирать только глобальные символы. Параметр <size> можно также задать именем символа и в этом случае размер определяется по завершению символа. Для фильтров filter и tracestop при отсутствии параметра <size> и указании параметра <start> символом размер определяется по завершению этого символа.
  Если параметр <size> не задан, а в качестве <start> указано *, начало и размер рассчитываются по первому и последнему символу, т. е. трассируется весь файл.
  Если представлены имена символов или *, перед ними и после них должны быть пробельные символы.
  Передаваемый ядру фильтр не обязательно совпадает с введённым. Для просмотра передаваемого фильтра используйте опцию -v.
  Ядро может оказаться не способным настроить область трассировки, если она не находится в одном отображении. Можно проверить события MMAP (или /proc/<pid>/maps) для контроля этой возможности.

[1]Debug with Arbitrary Record Format - отладка с произвольным форматом записи.
[2]Last Branch Record - запись последнего ветвления.
[3]Berkeley Packet Filter.

Фильтры могут разделяться пробелами или запятыми.

--exclude-perf

Не записывать события, вызванные самой программой perf. Опцию следует указывать после селектора событий (-e), который выбирает точки трассировки. Она дополняет фильтры выражением common_pid != $PERFPID. При наличии другой опции --filter новый фильтр будет объединяться с ней операцией И (&&).

-a, --all-cpus

Сбор данных в масштабе системы со всех CPU (принято по умолчанию, если цель не задана).

-p, --pid=

Запись событий для существующих процессов, заданных списком разделённых запятыми идентификаторов.

-t, --tid=

Запись событий для существующих потоков, заданных списком разделённых запятыми идентификаторов. Эта опция также запрещает наследование по умолчанию (включается опцией --inherit).

-u, --uid=

Записывает события в потоках, принадлежащих uid (имя или номер).

-r, --realtime=

Собирает данные с указанным приоритетом RT SCHED_FIFO.

--no-buffering

Собирает данные без буферизации.

-c, --count=

Период событий для выборки.

-o, --output=

Имя выходного файла.

-i, --no-inherit

Дочерние задачи не наследуют счётчики.

-F, --freq=

Профилирование с указанной частотой. Для использования максимально доступной частоты служит значение max (значение переменной kernel.perf_event_max_sample_rate). Будет снижать частоту выборки до максимальной частоты, разрешённой в данный момент (см. --strict-freq).

--strict-freq

Ведёт к отказу, если заданная частота не может применяться.

-m, --mmap-pages=

Число страниц данных mmap (должно быть степенью 2) или размер с суффиксом единицы (B/K/M/G). Размер округляется до ближайшей степени 2. Можно также указать через запятую число страниц mmap для области трассировки AUX.

--group

Помещать все события в одну группу. Предшествует опции --event и сохранена для совместимости с прежними версиями.

-g

Включает запись графа вызовов.

--call-graph

Организует и включает запись графа вызовов, подразумевая опцию -g. Можно выбрать режим fp (применяется по умолчанию), dwarf (DWARF CFI[1]) или lbr в качестве метода сбора информации для графа вызовов.

В некоторый системах, где при сборке применяется команда gcc --fomit-frame-pointer, метод fp, будет давать «фиктивный» граф вызовов и следует применять метод dwarf, если это возможно (модули perf собраны с библиотекой libunwind или libdw). Для метода lbr не требуются какие-либо опции компилятора и он будет строить граф по аппаратным регистрам LBR. Основным ограничением этого режима является его доступность лишь на новых платформах Intel, таких как Haswell. Он может давать лишь цепочку пользовательских вызовов и не работает одновременно в выборкой стека ветвлений.

При использовании метода dwarf программа perf записывает также дамп (пользовательского) стека при выборках. По умолчанию размер дампа стека составляет 8192 байта. Можно изменить размер, указав новое значение через запятую, например, --call-graph dwarf,4096.

-q, --quiet

Отключает вывод сообщений, что может быть полезно при использовании сценариев.

-v, --verbose

Задаёт более подробный вывод (ошибки открытия счётчиков и т. п.).

-s, --stat

Задаёт запись счётчиков событий по потокам (thread). Для просмотра значений применяется команда perf report -T.

-d, --data

Запись виртуальных адресов выборок.

--phys-data

Запись физических адресов выборки.

-T, --timestamp

Запись временных меток выборки. Для просмотра меток можно использовать perf report -D.

-P, --period

Запись периода выборки.

--sample-cpu

Запись CPU для выборки.

-n, --no-samples

Отказ от выборки.

-R, --raw-samples

Собирать необработанные записи выборки из всех открытых счётчиков принято по умолчанию для счётчиков точек трассировки).

-C, --cpu

Делать выборки лишь для заданных списком CPU. Процессоры указываются через запятую (0,1) или диапазоном (0-2) без пробелов. В режиме выборки по потокам при включённом наследовании принято по умолчанию), выборка выполняется лишь при выполнении потоков на указанных CPU. По умолчанию отслеживаются все процессоры.

[1]Call Frame Information - информация кадра вызова.

-B, --no-buildid

Не сохранять идентификаторы сборки двоичных файлов в perf.data. В результате пропускается постобработка после записи, которая иногда может быть долгой, поскольку требуется обрабатывать все события в поиске записей mmap. Недостатком этого подхода является возможность ошибок преобразования символов, если двоичные файлы задачи, использованной при записи, были обновлены или собраны заново, поскольку они доступны в этом случае лишь по именам. Можно также установить для конфигурационной переменной record.build-id значение skip для постоянного отказа от записи идентификаторов сборки.

-N, --no-buildid-cache

Не обновлять кэш идентификаторов сборки. Это снижает издержки обработки в некоторых ситуациях, когда достаточно информации в файле perf.data (включает buildid). Можно также отказаться от кэширования постоянно, указав в конфигурационном файле для переменной record.build-id значение no-cache.

-G name,..., --cgroup name,...

Задаёт отслеживание лишь контейнера (cgroup) с именем name. Опция доступна только в режиме отслеживания по процессорам (per-cpu). Должна быть смонтирована файловая система cgroup. Все потоки, относящиеся к контейнеру name, отслеживаются при работе на контролируемых CPU. Можно указать несколько контейнеров cgroup, каждый из которых будет применяться к соответствующему событию (первое событие для первого контейнера cgroup, второе - для второго и т. д.). Можно указать пустой контейнер cgroup (отслеживать всегда), например, -G foo,,bar. Контейнеры cgroup должны иметь соответствующие события, т. е. всегда указывать на событие, указанное ранее в командной строке. Если нужно отслеживать несколько событий в одном контейнере cgroup, можно указать -e e1 -e e2 -G foo,foo или просто -e e1 -e e2 -G foo.

Если нужно отслеживать, например, циклы для cgroup в масштабе всей системы, можно воспользоваться командой perf stat -e cycles -G cgroup_name -a -e cycles.

-b, --branch-any

Включает выборку стека ветвления. Могут выбираться любые типы ветвлений. Эта опция является сокращением для --branch-filter any (см. --branch-filter).

-j, --branch-filter

Включает выборку стека ветвления. Каждая выборка фиксирует цепочку последовательных ветвлений, число которых зависит от оборудования, типа интересующих ветвлений и исполняемого кода. Можно выбирать типы фиксируемых ветвлений, включив фильтр. Поддерживаемые фильтры включают:

- any - все типы ветвлений;
- any_call - вызов любой функции или системный вызов;
- any_ret - возврат из любой функции или системного вызова;
- ind_call - любое непрямое ветвление;
- call - прямые вызовы включая длинные (в ядро или из него);
- u - только ветвления на пользовательский уровень;
- k - только ветвления на уровень ядра;
- hv - только ветвления на уровень гипервизора;
- in_tx - только ветвления в аппаратные транзакции;
- no_tx - кроме ветвлений в аппаратные транзакции;
- abort_tx - только ветвления для прерываний аппаратных транзакций;
- cond - условные ветвления;
- save_type - сохранять тип ветвления при выборке, если двоичный код позднее не доступен.

Опция требует хотя бы один тип ветвления из any, any_call, any_ret, ind_call, cond. Уровни привилегий можно опускать и в этом случае для фильтра ветвления используется уровень привилегий связанных событий. Использоваться могут уровни привилегий ядра (k) и гипервизора (hv). При выборке множества событий включается выборка стека ветвления для всех отслеживаемых событий. Тип выбираемого ветвления совпадает для всех событий. Разные фильтры должны указываться через запятые, например, --branch-filter any_ret,u,k. Фильтрация ветвления доступна не для всех процессоров.

--weight

Включает взвешенную выборку. Записывается дополнительно вес каждой выборки, который можно просматривать с помощью ключей сортировки weight и local_weight. В настоящее время это работает для событий прерывания TSX1 и некоторых событий памяти в режиме precise на современных Intel CPU.

--namespaces

Запись событий типа PERF_RECORD_NAMESPACES.

--transaction

Запись флагов транзакции для связанных с транзакциями событий **.**

--per-thread

Использование mmap на уровне потока (thread). По умолчанию mmap создаются на уровне процессора, но данная опция позволяет переопределить это поведение. Побочным эффектом является запрет наследования. Опция --per-thread игнорируется с выдачей предупреждения при наличии в команде опции -a или -C.

-D, --delay=

Задаёт задержку начала измерений после запуска программы (в миллисекундах). Это полезно для исключения измерений на стартовой фазе программы, когда её поведение отличается.

-I, --intr-regs

Фиксация состояния машины (регистров) в момент прерывания, т. е. при переполнении счётчика для каждой выборки. Набор регистров зависит от архитектуры. Опция по умолчанию отключена. Можно указать регистры для выборки по их символьным именами, например, для x86 - ax, si. Для получения списка доступных регистров используется --intr-regs=\?. Регистры указываются через запятые, например, --intr-regs=ax,bx.

--user-regs

Похожа на -I, но фиксирует в момент выборки пользовательские регистры. Список регистров можно получить с помощью --user-regs=\?.

--running-time

Запись времени работы и активности для чтения событий (:S).

-k, --clockid

Задаёт идентификатор часов для использования в полях времени различных записей perf_event_type (см. clock_gettime()). Поддерживаются, в частности, часы CLOCK_MONOTONIC и CLOCK_MONOTONIC_RAW, а некоторые события могут использовать CLOCK_BOOTTIME, CLOCK_REALTIME и CLOCK_TAI.

-S, --snapshot

Выбирает режим Snapshot с областью AUX. Опция действует только с событиями трассировки области AUX. Дополнительно можно указать число байтов на снимок. В режиме Snapshot данные трассировки фиксируются лишь при получении сигнала SIGUSR2.

--proc-map-timeout

Обработка имеющихся потоков /proc/XXX/mmap может занять много времени по причине очень больших файлов. В таких случаях нужно задать тайм-аут, который устанавливается это опцией (по умолчанию 500 мсек).

--switch-events

Запись событий переключения контекста (PERF_RECORD_SWITCH или PERF_RECORD_SWITCH_CPU_WIDE).

--clang-path=PATH

Путь к библиотеке clang для использования при компиляции сценариев BPF (включается при поддержке BPF).

--clang-opt=OPTIONS

Опции, передаваемые clang при компиляции сценариев BPF (включается при поддержке BPF).

--vmlinux=PATH

Путь к файлу vmlinux с отладочной информацией (включается при поддержке BPF).

--buildid-all

Запись идентификаторов сборки для всех DSO, независимо от их участия.

--aio[=n]

Использовать <n> блоков управления в асинхронном режиме записи трассировки (Posix AIO) (по умолчанию 1, максимально 4). Асинхронный режим поддерживается лишь при компоновке perf с библиотекой libc, обеспечивающей реализацию Posix AIO API.

--affinity=mode

Задаёт маску сходства потоков чтения трассировки в соответствии с политикой, заданной значением mode. Режим node (узел) задаёт маску сходства по маске узла NUMA процессора, обрабатывающего буфер mmap, cpu задаёт маску сходства по процессору, обрабатывающему буфер mmap.

--mmap-flush=number

Задаёт минимальное число байтов, извлекаемых из страниц данных mmap и обрабатываемых для вывода. Можно использовать суффиксы B/K/M/G (байты, кило-, мега- и гигабайты).

Максимально разрешённое значение составляет четверть от размера страниц mmap.

По умолчанию используется значение 1 байт, которое указывает, что при каждом обнаружении потоком записи вывода новых данных в буфере mmap эти данные извлекаются (возможно со сжатием при -z) и записываются в файл perf.data или конвейер.

Более крупные блоки данных сжимаются эффективней мелких, поэтому извлечение крупных блоков страниц данных mmap является предпочтительным с точки зрения выходного размера.

В некоторых случаях снижение числа системных вызовов записи вывода за счёт большего размера блоков может уменьшать время выполнения и издержки профилирования.

-z, --compression-level[=n]

Задаёт сжатие трассировки с использованием уровня n (по умолчанию 1 - максимальная скорость, для максимального сжатия используется уровень 22).

--all-kernel

Задаёт для всех используемых событий работу в пространстве ядра.

--all-user

Задаёт для всех используемых событий работу в пользовательском пространстве.

--timestamp-filename

Добавляет временную метку к выводимому имени файла.

--timestamp-boundary

Записывает границы временных меток (время первой и последней выборки).

--switch-output[=mode]

Задаёт создание множества файлов perf.data с временной меткой в качестве префикса, переходя к новому файлу в соответствии с режимом. В режиме signal (используется по умолчанию) переход происходит по получению SIGUSR2, в режиме <size> - по достижению заданного размера[1] (число с суффиксом B/K/M/G), <time> - по истечении заданного времени (число с суффиксом s/m/h/d).

Возможным вариантом использования является «нарезка» файлов perf.data для последующей обработки (возможно в perf script) с возможностью отказа от сохранения следующего файла.

Предполагаются опции --timestamp-filename, --no-buildid и --no-buildid-cache. Две последних опции служат для снижения издержек, связанных с переходом к другому файлу. Их можно выключить с помощью

```
--switch-output --no-no-buildid --no-no-buildid-cache
```

--switch-max-files=N

При ротации perf.data с опцией --switch-output задаёт сохранение только N файлов.

--dry-run

Разбор опций и завершение работы. Служит для обнаружения ошибок в командах. Команда perf record --dry-run -e может применяться для компиляции сценариев BPF, если в конфигурационном файле указано значение true для переменной llvm.dump-obj.

--tail-synthesize

Вместо сбора неотбираемых событий (например, fork, comm, mmap) в начале записи задаёт их сбор при завершении выходного файла. Эти события отражают состояние системы при завершении записи.

--overwrite

Заставляет использовать перезаписываемый кольцевой буфер для всех событий. При заполнении буфера продолжается запись в начало и первые события не попадут в файл perf.data.

При использовании опций --overwrite и --switch-output программа perf записывает и отбрасывает события до получения сигнала, означающего необычное событие, требующее создания моментального снимка (snapshot) самых последних событий, которые в этот момент находятся в кольцевом буфере.

Атрибут overwrite можно установить или сбросить с помощью конфигурации. Например, cycles/overwrite/ и instructions/no-overwrite/.

Предполагается опция --tail-synthesize.

[1]Точность перехода по значению размера сильно зависит от конфигурации - числа и размера кольцевых буферов (-m). При больших размерах файлов (> 5 Мбайт) точность возрастает.

report

Считывает файл данных perf.data, созданный командой perf record , и выводит профиль производительности, построенный по собранным ранее данным.

Синтаксис

```
perf report [-i <file> | --input=file]
```

Опции

-i, --input=
Имя входного файла (по умолчанию perf.data, если stdin не является fifo).

-v, --verbose
Задаёт подробный вывод (адреса символов и т. п.)

-q, --quiet
Отменяет вывод сообщения, отменяя опцию -v.

-n, --show-nr-samples
Выводит число выборок для каждого символа.

--show-cpu-utilization
Показывает загрузку процессора для разных режимов.

-T, --threads
Показывает счётчики событий для потоков (thread). Фал данных для этого следует записывать с опцией -s.

-c, --comms=
Задаёт учёт символов лишь в указанных столбцах. Можно использовать файл CSV, заданный параметром file://filename. Эта опция будет влиять на столбец overhead (см. --percentage).

--pid=
Задаёт вывод событий лишь для указанного идентификатора процесса (список разделённых запятыми значений).

--tid=
Задаёт вывод событий лишь для указанного идентификатора потока (список разделённых запятыми значений).

-d, --dsos=
Задаёт учёт лишь символов указанных dso. Можно использовать файл CSV, заданный параметром file://filename. Эта опция будет влиять на столбец overhead (см. --percentage).

-S, --symbols=
Задаёт учёт лишь указанных символов. Можно использовать файл CSV, заданный параметром file://filename. Эта опция будет влиять на столбец overhead (см. --percentage).

--symbol-filter=
Задаёт вывод лишь символов, которые соответствуют (частично) данному фильтру.

-U, --hide-unresolved
Задаёт вывод лишь записей, преобразованных в символы.

-s, --sort=
Задаёт сортировку гистограммы по указанным ключам. Для задания множества ключей можно использовать формат CSV. Доступна сортировка по ключам pid, comm, dso, symbol, parent, cpu, socket, srcline, weight, local_weight, cgroup_id.
Значения ключей описаны ниже.

- comm - команда (имя) задачи, которую можно прочитать в /proc/<pid>/comm.
- pid - команда и идентификатор потока для задачи.
- dso - имя библиотеки или модуля, выполняемого в момент выборки.
- dso_size - размер библиотеки или модуля, выполняемого в момент выборки.
- symbol - имя функции, выполняемой в момент выборки.
- symbol_size - размер функции, выполняемой в момент выборки.
- parent - имя функции, соответствующей регулярному выражению родительского фильтра. Не соответствующие записи выводятся как [other].
- cpu - номер cpu задачи, выполнявшейся в момент выборки.
- socket - номер процессорного сокета задачи, выполнявшейся в момент выборки.
- srcline - имя файла номер строки, выполнявшейся в момент выборки. Должна быть предоставлена отладочная информация DWARF.
- srcfile - имя файла совпадает с именем файла исходного кода. Требуются данные DWARF.
- weight - связанный с событием (глобальный) вес, например, задержка памяти или стоимость прерывания транзакции.
- local_weight - локальный вариант указанного выше веса.
- cgroup_id - идентификатор, выведенный из значений пространства имён cgroup для устройства и inode.
- transaction - флаги прерывания транзакции.
- overhead - процент издержек для выборки.
- overhead_sys - процент издержек для выборки в системном режиме.
- overhead_us - процент издержек для выборки в пользовательском режиме.
- overhead_guest_sys - процент издержек для выборки в системном режиме на гостевой машине.
- overhead_guest_us - процент издержек для выборки в пользовательском режиме на гостевой машине.
- sample - номер выборки.
- period - необработанное значение счётчика событий для выборки.
- time - разделяет выборки по временным меткам с разрешением, заданным параметром --time-quantum (по умолчанию 100 мсек). Указывается с издержками или перед ними.

По умолчанию применяются ключи comm, dso и symbol (т. е. --sort comm,dso,symbol). Если задана опция --branch-stack, доступны также перечисленные ниже ключи.

- dso_from - имя библиотеки или модуля, где произошло ветвление.
- dso_to - имя библиотеки или модуля, куда было выполнено ветвление.
- symbol_from - имя функции, где произошло ветвление.
- symbol_to - имя функции, куда было выполнено ветвление.
- srcline_from - файл исходного кода и строка, где произошло ветвление.
- srcline_to - файл исходного кода и строка, куда было выполнено ветвление.

- mispredict - N для предсказанного ветвления, Y - для ошибочно предсказанного.
- in_tx - ветвление в транзакции TSX[1].
- abort - прерывание транзакции TSX.
- cycles - циклы в базовом блоке.

Принятые по умолчанию ключи сортировки заменяются на comm, dso_from, symbol_from, dso_to и symbol_to (см. --branch-stack).

Когда заданы ключи сортировки, столбцы IPC[2] и IPC Coverage включаются автоматически. IPC показывает среднее значение на функцию, а IPC coverage - процент инструкций, которые отобраны в этой функции. Низкое значение говорит о возможном узком месте при выполнении функции, например, при доступе к памяти. Если функция имеет высокую загрузку (overhead) и малое значение IPC, следует дополнительно проанализировать её для оптимизации.

При использовании опции --mem-mode доступны также перечисленные ниже ключи (несовместимо с --branch-stack).

- symbol_daddr - имя символа данных, выполняемого в момент выборки.
- dso_daddr - имя библиотеки или модуля, содержащего данные, выполняемые в момент выборки.
- locked - была ли шина заблокирована в момент выборки.
- tlb - тип tlb для доступа к данным в момент выборки.
- mem - тип доступа к памяти для данных в момент выборки.
- snoop - тип snoop (если есть) для данных в момент выборки.
- dcacheline - адрес данных cacheline в момент выборки.
- phys_daddr - физический адрес данных, выполняемых в момент выборки.

Принятые по умолчанию ключи сортировки заменяются на local_weight, mem, sym, dso, symbol_daddr, dso_daddr, snoop, tlb, locked (см. --mem-mode).

Если файл данных имеет события точек трассировки, доступны также приведённые ниже (динамические) ключи сортировки.

- trace - вывод трассировки в один столбец.
- trace_fields - вывод трассировки в отдельных столбцах.
- [<event>.]<field>[/raw] - необязательное имя события и имя указанного поля

Последняя форма включает имена события и поля. Если имя события опущено, выполняется поиск всех событий, соответствующий имени поля. Совпадающее поле будет выводиться только для событий, имеющих это поле. Для имён событий поддерживается сопоставление подстроки, поэтому пользователь не обязан каждый раз полностью указывать подсистему и имя события. Например, sched:sched_switch можно сократить для switch, если не возникает неоднозначности. Событие также можно указать индексом (отсчёт с 1) с префиксом % (%1 для первого события, %2 для второго и т. д.).

Имя поля может иметь суффикс /raw, который отключает «красивую» печать и имя поля выводится подобно шестнадцатеричным числам. Опция --raw-trace будет давать одинаковый эффект для всех динамических ключей сортировки.

Используемый по умолчанию ключ сортировки меняется на trace, если все события в файле данных являются точками трассировки.

-F, --fields=

Задаёт поля вывода, позволяя указать множество ключей в формате CSV. Доступны поля overhead, overhead_sys, overhead_us, overhead_children, sample и period. Можно также указывать любые ключи сортировки. По умолчанию ключи сортировки, не заданные в -F, добавляются автоматически. Если ключи начинаются с префикса +, указанные поля будут добавлены после используемых по умолчанию полей. Например, perf report -F +period,sample.

-p, --parent=<regex>

Регулярное выражение для идентификации родителя, каковым является вызвавший эту функцию. Поиск родителя выполняется в цепочке вызовов, поэтому нужно записать эту информацию. Шаблон поиска использует расширенный формат регулярных выражений. По умолчанию используется ^sys_|^do_page_fault (см. --sort parent).

-x, --exclude-other

Задаёт вывод лишь записей для соответствующего родителя.

-w, --column-widths=<width[,width...]>

Задаёт ширину каждого столбца в выводе для удобочитаемости. Используемое по умолчанию значение 0 задаёт неограниченную ширину.

-t, --field-separator=

Задаёт указанный символ в качестве разделителя полей взамен пробелов с заменой этого символа в именах и других полях точкой (которая поэтому не может служить разделителем).

-D, --dump-raw-trace

Дамп необработанной трассировки в формате ASCII.

-g, --call-graph=<print_type,threshold[,print_limit],order,sort_key[,branch],value>

Отображение цепочек вызовов с использованием print_type, threshold, print_limit, order, sort_key, (необязательно) branch и value. Порядок не фиксирован и параметры можно размещать произвольно. Исключением является лишь размещение print_limit после threshold.

Параметр print_type может принимать любое из приведённых ниже значений.

- flat - один столбец, линейное отображение всех цепочек вызовов.
- graph - дерево графа с выводом абсолютных значений издержек (принято по умолчанию).
- fractal - похоже на graph, но выводятся относительные значения. Каждая ветвь дерева считается новым объектом профилирования.
- folded - цепочки вызовов отображаются в строке с разделением точкой с запятой (;).
- none - цепочки вызовов не отображаются.

Параметр threshold задаёт минимальное процентное значение, которое включается в вывод графа вызовов (по умолчанию 0,5 %).

Параметр print_limit применяется лишь для интерфейса stdio и ограничивает число записей вызовов в одной записи гистограммы. Отметим, что его нужно указывать после параметра threshold (не обязательно сразу). По умолчанию ограничений не задано (0).

Параметр order может принимать одно из значений:

[1]Transactional Synchronization Extension - расширение синхронизации транзакций.
[2]Instruction Per Cycle - число инструкций на цикл.

- callee - граф вызовов по вызываемым;
- caller - обращённый граф вызовов по вызывающим.

По умолчанию используется caller, если задана опция --children, иначе callee.

Параметр sort_key может принимать значения:

- function - по функциям (принято по умолчанию);
- address - по индивидуальным адресам кода;
- srcline - по файлам исходного кода и номерам строк.

Необязательный параметр branch задаёт включение информации о ветвлении (если она доступна) в граф вызовов. Обычно удобней для этого использовать опцию --branch-history.

Параметр value может принимать значения:

- percent - вывод издержек в процентах (принято по умолчанию);
- period - вывод периода события;
- count - вывод счётчика события.

--children

Задаёт накопление дочерних вызовов родительской записи для последующего отображения. Вывод будет включать столбец Children, по которому будет выполняться сортировка данных. Для этого требуется запись цепочек вызовов. Дополнительные сведение приведены в разделе «Расчёт издержек». Опция по умолчанию включена, отключить её можно опцией --no-children.

--max-stack

Задаёт ограничение глубины стека при разборе цепочки вызовов. Это задаёт компромисс между объёмом информации и скоростью обработки для заданий, имеющих очень большую глубину стека вызовов. Отметим, что при использовании опции --itrace синтезированный размер цепочки вызовов будет переопределять это значение, если оно меньше синтезированного. По умолчанию используется глубина стека 127.

-G, --inverted

Псевдоним для обращённого графа по вызывающим.

--ignore-callees=<regex>

Задаёт игнорирование вызываемых (callee) для функций, соответствующих регулярному выражению regex. Это влияет на сбор данных о вызывающих для каждой такой функции в дереве графа вызовов.

--pretty=<key>

Задаёт «красивый» стиль вывода. Ключ может принимать значение normal или raw.

--stdio

Задаёт использование интерфейса stdio.

--stdio-color

Значения always (всегда), never (никогда) или auto (автоматически) управляют выводом цвета через командную строку в дополнение к параметрам color.ui в конфигурационном файле. Опцию --stdio-color следует всегда использовать для управления цветом при перенаправлении вывода в конвейер или файл. Опция --stdio-color без параметра задаёт использование цвета всегда.

--tui

Задаёт использование интерфейса TUI, интегрированного с аннотированием и позволяющего масштабировать (zoom) вывод с просмотром DSO или потоков (thread), а также обеспечивающего другие возможности. Опция --tui требует наличия терминала и при его отсутствии или использовании конвейера с другими командами будет применяться интерфейс stdio.

--gtk

Задаёт использование интерфейса GTK2.

-k, --vmlinux=<file>

Путь к файлу vmlinux.

--ignore-vmlinux

Задаёт игнорирование файлов vmlinux.

--kallsyms=<file>

Путь к файлу kallsyms.

-m, --modules

Задаёт загрузку символов модулей. Важно отметить, что это следует применять с опцией -k и «живым» ядром.

-f, --force

Отключает проверку владения.

--symfs=<directory>

Задаёт поиск файлов с символами относительно указанного каталога.

-C, --cpu

Задаёт включение в отчёт лишь выборок для указанных списком CPU. Процессоры можно перечислить через запятые (0,1) или указать диапазон (0-2) без пробелов. По умолчанию учитываются все CPU.

-M, --disassembler-style=

Задаёт стиль дизассемблирования для objdump.

--source

Задаёт чередование исходного кода с ассемблерным. По умолчанию включено, отключается опцией --no-source.

--asm-raw

Задаёт вывод необработанного кодирования ассемблерных инструкций.

--show-total-period

Задаёт вывод столбца с суммой периодов.

-I, --show-info

Задаёт вывод расширенной информации о файле perf.data. Эти данные могут быть очень велики и занять много места на экране. В настоящее время они включают топологию cpu и numa для хост-системы.

-b, --branch-stack

Использовать для построения гистограммы адреса ветвлений в выборках вместо адресов инструкций. Для создания значимого вывода файл perf.data должен быть записан с использованием опции record -b или perf record --branch-filter xxx, где xxx задаёт фильтр ветвления. Команда perf report может автоматически определять наличие стеков ветвления в файле perf.data и переключаться в режим просмотра ветвлений, если не задана опция.

--branch-history

Задаёт добавление адресов ветвлений в выборках к стеку вызовов. Это позволяет увидеть путь, который проходит программа при каждой выборке. Данные должны собираться с использованием опций -b (или -j) и -g.

--objdump=<path>

Путь к двоичному файлу objdump.

--group

Задаёт совместный вывод информации для группы событий, форсируя групповой вывод, если группы не были определены в файле данных.

--demangle

Переводит имена символов в удобочитаемый формат. Опция включена по умолчанию и отключается с помощью --no-demangle.

--demangle-kernel

Переводит имена символов ядра в удобочитаемый формат (для ядер C++).

--mem-mode

Задаёт использование адресов данных в выборках в дополнение к адресам инструкций при построении гистограмм. Для создания значимого вывода файл perf.data должен быть записан с помощью команды perf record -d -W и с указанием специального события -e cpu/mem-loads/p или -e cpu/mem-stores/p (см perf mem).

--percent-limit

Отключает вывод записей с издержками ниже заданного порога (по умолчанию 0). Эта опция также устанавливает порог для цепочек вызовов, однако используемое для этого порога значение по умолчанию отличается от значения, применяемого по умолчанию для гистограмм (см. --call-graph).

--percentage

Задаёт способ отображения процента издержек для отфильтрованных записей. Фильтры можно задать опциями --comms, --dsos, --symbols или операциями Zoom в TUI (thread, dso и т. п.).

Значение relative задаёт вывод относительно отфильтрованных записей и сумма всегда будет составлять 100%. В случае absolute расчёт выполняется относительно исходных значений до и после фильтрации.

--header

Задаёт вывод заголовка файла perf.data, включающего такую информацию, как имя хоста, версии ОС и perf, данные процессоров и памяти, командная строка perf, список событий и т. п. В настоящее время эта функция поддерживается только для режима вывода --stdio.

--header-only

Задаёт вывод лишь заголовка perf.data (включает опцию --stdio).

--time

Задаёт анализ выборок лишь из временного окна <start>,<stop>. Время указывается в формате секунды.микросекунды. Если время начала не указано (т. е. ,x.y), анализ начинается с начала файла. Если время окончания не указано (т. е. x.y,), анализ выполняется до конца файла.

Поддерживается также указание интервалов в процентах - a%/n,b%/m,... или a%-b%,c%-%d,...

Например, для выбора вторых 10% интервала записи можно использовать команду

```
perf report --time 10%/2
```

Для выбора интервала от 0% до 10% служит команда

```
perf report --time 0%-10%
```

Например, для выбора первых и вторых 10% интервала записи можно использовать команду

```
perf report --time 10%/1,10%/2
```

Для выбора интервалов от 0% до 10% и от 30% до 40% служит команда

```
perf report --time 0%-10%,30%-40%
```

--itrace

Задаёт опции декодирования данных трассировки инструкций:

- i синтезировать события инструкций;
- b синтезировать события ветвления;
- c синтезировать события ветвления (только вызовы);
- r синтезировать события ветвления (только возвраты);
- x синтезировать события транзакций;
- w синтезировать события ptwrite;
- p синтезировать события, связанные с питанием;
- e синтезировать события, связанные с ошибками;
- d создать журнал отладки;
- g синтезировать цепочку вызовов (используется с i или x);
- l синтезировать записи последнего ветвления (используется с i или x);
- s пропустить указанное число записей в начале.

По умолчанию учитываются все события (--itrace=ibxwpe), за исключением perf (--itrace=ce)

В дополнение к этому можно задать интервал учёта (по умолчанию 100000, за исключением perf script, где устанавливается 1) в удобных единицах:

- i инструкции;
- t такты часов;
- ms миллисекунды;
- us микросекунды;
- ns наносекунды (используется по умолчанию).

Можно также указать размер цепочки (по умолчанию 16, максимум 1024) для событий.

Можно указать также число записей последнего ветвления (по умолчанию 64, максимум 1024).

Можно пропустить генерацию событий (инструкции, ветвления, транзакции, ptwrite, питание) в начале. Это удобно для пропуска фазы инициализации. Например, опция

```
--itrace=i0nss1000000
```

обеспечит пропуск первого миллиона инструкций.

Для полного запрета декодирования служит опция --no-itrace.

--full-source-path

Выводит полный путь к файлам исходного кода при выводе номеров строк.

--show-ref-call-graph
При выборке нескольких событий может не требоваться сбор графа вызовов для всех этих событий. Точки выборки обычно расположены близко и достаточно собрать графы вызовов для опорных событий. Поэтому можно использовать модификатор событий call-graph=no для запрета графа вызовов других событий с целью снижения нагрузки. Эта опция позволяет команде perf report показать опорные графы вызовов без сбора всех графов.

--socket-filter
Выводить только выборки для процессорного сокета, соответствующего фильтру.

--samples=N
Сохранять N отдельных выборок для каждой записи гистограммы с целью показа контекста в браузере perf report TUI.

--raw-trace
При выводе трассировки событий не применять форматирование вывода и плагины.

--hierarchy
Включает иерархический вывод.

--inline
Если адрес графа вызовов относится к inline-функции, будет выводиться inline-стек, каждая запись которого указывает имя функции или файл и строку. Опция включена по умолчанию, отключается с помощью -no-inline.

--mmaps
Показывает вывод --tasks и информацию mmap в формате, похожем на /proc/<PID>/maps.
Отметим, что сохраняются не все mmap.

--ns
Вывод временных меток в наносекундах.

--stats
Выводит суммарную статистику событий без дополнительной обработки (похоже на завершение вывода по команде perf report -D).

--tasks
Выводит отслеживаемые задачи, сохранённые в файле perf.data, с указанием pid/tid/ppid и командной строки. Вывод организован так, чтобы можно было различить родительские и дочерние задачи.

--percent-type
`Задаёт тип расчёта процентов – global-period (за период по всем данным), local-period (за период в области действия функции), global-hits (от числа выбором по всем данным) или local-hits (от числа выбором в области действия функции).`

--time-quantum
Задаёт величину квантования для ключа сортировки по времени (по умолчанию 100 мсек). Можно задать время в секундах, милли-, микро- и наносекундах.

Расчёт издержек
См. Расчёт издержек в описании top.

annotate
Читает файл perf.data (создаётся с помощью perf record) и выводит аннотированный код. Если объектный файл включает отладочные символы, ассемблерный код будет сопровождаться исходным кодом. При отсутствии отладочной информации выводится лишь аннотированный ассемблерный код.

Синтаксис

```
perf annotate [-i <file> | --input=file] [symbol_name]
```

Опции
-i, --input=<file>
Имя входного файла (по умолчанию perf.data, если stdin не является fifo).

-d, --dsos=<dso[,dso...]>
Задаёт учёт лишь указанных dso.

-s, --symbol=<symbol>
Символ для аннотирования.

-f, --force
Отмена проверки владения.

-v, --verbose
Подробный вывод (адреса символов и т. п.).

-q, --quiet
Не показывать сообщений (отменяет -v).

-n, --show-nr-samples
Показывать число выборок для каждого символа.

-D, --dump-raw-trace
Дамп необработанной трассировки в формате ASCII.

-k, --vmlinux=<file>
Путь к файлу vmlinux.

--ignore-vmlinux
Игнорировать файлы vmlinux.

-m, --modules
Загружать символы модулей (используется только с -k и «живым» ядром).

-l, --print-line
Выводить соответствующие строки исходного кода (может быть медленно).

-P, --full-paths
Не сокращать выводимые пути.

--stdio
Использовать интерфейс stdio.

--stdio2
Использовать интерфейс stdio2 с форматированием TUI.

--stdio-color=<mode>
Управляет использованием цвета через командную строку - always (всегда), never (никогда) или auto (автоматически) в дополнение к конфигурационному параметру color.ui. Опция --stdio-color позволяет использовать выделение цветом даже при выводе в конвейер или файл. Опция --stdio-color задаёт использование цвета всегда.

--tui
Использовать интерфейс TUI. Для этой опции нужен терминал. Если терминала нет или команда применяется в конвейере, будет использован интерфейс stdio.

--gtk
Использовать интерфейс GTK.

-C, --cpu=<cpu>
Сообщать только о выборках для указанных CPU. Процессоры можно указать списком через запятые (0,1) или диапазоном (0-2) без пробелов. По умолчанию учитываются все CPU.

--asm-raw
Показывать необработанное представление ассемблерных инструкций.

--show-total-period
Показывать столбец с суммой периодов.

--source
Чередовать исходный код с ассемблерным. По умолчанию включено, отключается опцией --no-source.

--symfs=<directory>
Поиск файлов с символами относительно указанного каталога.

-M, --disassembler-style=
Устанавливает стиль дизассемблера для objdump.

--objdump=<path>
Путь к библиотеке objdump.

--skip-missing
Пропускать символы, которые не могут быть аннотированы.

--group
Показывать информацию группы событий вместе.

--percent-type
Задаёт тип расчёта процентов - global-period (за период по всем данным), local-period (за период в области действия функции), global-hits (от числа выбором по всем данным) или local-hits (от числа выбором в области действия функции).

ftrace

Простая оболочка для работы с функциональностью ядра ftrace. В настоящее время поддерживается лишь трассировка одного потока и просто записывается вывод trace_pipe в текстовом формате на stdout.

Синтаксис

```
perf ftrace <command>
```

Опции

-t, --tracer=
Используемый трассировщик (function_graph или function).

-v, --verbose=
Уровень детализации.

-p, --pid=
трассировка процессов с указанными идентификаторами (через запятые).

-a, --all-cpus
Сбор данных в масштабе системы. При работе без параметра <command> опция -a включается по умолчанию, в остальных случаях её нужно задавать явно.

-C, --cpu=
Выполнять трассировку только для указанных CPU. Процессоры можно указать списком через запятые (0,1) или диапазоном (0-2) без пробелов. По умолчанию учитываются все CPU.

-T, --trace-funcs=
Трассировать только указанные параметром функции. Опция может указываться несколько раз и разрешено использование регулярных выражений (glob). Функции передаются set_ftrace_filter в файловой системе tracefs.

-N, --notrace-funcs=
Не трассировать указанные параметром функции. Опция может указываться несколько раз и разрешено использование регулярных выражений (glob). Функции передаются set_ftrace_notrace в файловой системе tracefs.

-G, --graph-funcs=
Задаёт фильтр графа функций (или шаблон glob). Это полезно лишь для трассировщика function_graph и включает трассировку функций, вызываемых из указанной функции. Опцию можно применять многократно и она передаётся set_graph_function в файловой системе tracefs.

-g, --nograph-funcs=
Отключает граф указанных функций (или шаблон glob). Это полезно лишь для трассировщика function_graph и выключает трассировку функций, вызываемых из указанной функции. Опцию можно применять многократно и она передаётся set_graph_notrace в файловой системе tracefs.

-D, --graph-depth=
Задаёт максимальную глубину для трассировщика function_graph.

trace

Команда выводит события, связанные с целью трассировки. Изначально это вызовы syscall, но могут выводиться и другие события, такие как отказы страниц, время жизни задач, события планировщиков и т. п.

Эта программа может работать «вживую», а не только с файлами perf.data как другие инструменты perf. Файлы для анализа можно создавать с помощью perf record, но сеанс записи должен включать события raw_syscalls (-e

raw_syscalls:*). Можно использовать команду perf trace record, которая автоматически включает отслеживание событий raw_syscalls.

Синтаксис

```
perf trace
perf trace record
```

Опции

Ниже перечислены опции команды perf trace, а опции perf trace record можно найти в описании perf record.

-a, --all-cpus
Сбор данных в масштабе системы со всех CPU.

-e, --expr, --event
Список syscall и других событий perf (точки трассировки, события аппаратного кэша и т. п.) для показа. Поддерживаются шаблоны, например, epoll_*, msg и т. п. Полный список событий можно посмотреть по команде perf list. Префикс ! будет обеспечивать вывод всех syscall, кроме указанных. Для этого символа может потребоваться использование escape.

-D msecs, --delay msecs
Задаёт ожидание в течение указанного числа миллисекунд перед началом измерения после запуска команды. Это полезно для исключения переходных процессов, происходящих при старте.

-o, --output=
Имя выходного файла.

-p, --pid=
Запись событий имеющихся процессов с указанными через запятую идентификаторами.

-t, --tid=
Запись событий имеющихся потоков с указанными через запятую идентификаторами.

-u, --uid=
Запись событий из потоков, принадлежащих uid (имя или номер).

-G, --cgroup
Запись событий в потоках cgroup.
Группы cgroup для указания можно посмотреть в каталоге /sys/fs/cgroup/perf_event и указать нужную группу (A) в команде

```
perf trace -G A -e sched:*switch
```

Это установит все raw_syscalls:sys_{enter,exit}, pgfault, vfs_getname и т. п., а также _and_ sched:sched_switch в cgroup A, тогда как команда

```
perf trace -e sched:*switch -G A
```

установит только событие sched:sched_switch в группе A, а все другие события (raw_syscalls:sys_{enter,exit} и т. п.) останутся «без» cgroup (в корневой cgroup, на системном уровне и т. п.).
Можно задать несколько cgroup

```
perf trace -G A -e sched:*switch -G B
```

В этом случае syscall будут в группе A, sched:sched_switch - в группе B.

--filter-pids=
Отфильтровывает события указанных через запятую pid и самой трассировки.

-v, --verbose=
Задаёт уровень детализации вывода.

--no-inherit
Дочерние задачи не наследуют счётчики.

-m, --mmap-pages=
Число страниц данных mmap (должно быть степенью 2) или размер с суффиксом B/K/M/G (округляется до числа страниц, равного ближайшей степени 2).

-C, --cpu
Выполнять трассировку только для указанных CPU. Процессоры можно указать списком через запятые (0,1) или диапазоном (0-2) без пробелов. В режиме по потокам (per-thread) со включённым наследованием (принято по умолчанию), события фиксируются только при выполнении на назначенных CPU. По умолчанию учитываются все CPU.

--duration
Показывать только события с продолжительностью больше N.M мсек.

--sched
Накапливать поток в процессе выполнения и выводить результат в конце сессии.

--failure
Показывать только вызовы syscall с отказами (отличный от 0 код возврата).

-i, --input
Обработка событий из указанного файла perf.data.

-T, --time
Печатать полные временные метки, а не смещение от первой выборки.

--comm
Выводить процессы с идентификаторами. Принято по умолчанию, отключается опцией --no-comm.

-s, --summary
Показывать только сводку syscall с минимальными, максимальными и средними значениями в миллисекундах, а также stddev.

-S, --with-summary
Показывать только сводку syscall со сводкой по потокам с минимальными, максимальными и средними значениями в миллисекундах, а также stddev.

--tool_stats
Показывать статистику инструмента, такую как число обнаружений fd→pathname путём перехвата возвращаемого syscall значения + vfs_getname или путём чтения /proc/pid/fd и т. п.

-f, --force
Выполнять без лишних вопросов.

-F=[all|min|maj], --pf=[all|min|maj]
Трассировка отказов страниц. Можно указать второстепенные, важные или все отказы. По умолчанию важные (maj).

--syscalls
Трассировка системных вызовов. Опция включена по умолчанию, отключается с помощью --no-syscalls.

--call-graph [mode,type,min[,limit],order[,key][,branch]]
Организует и включает запись графа вызовов. Более подробное описание приведено для одноимённой опции perf record и perf report. Наиболее полезными для perf trace являются режимы dwarf и lbr, если они поддерживаются.
Использование опции от имени root user увеличивает значение --mmap-pages до 4 максимальных значения для других пользователей (в соответствии с переменной kernel.perf_event_mlock_kb). Это происходит лишь в тех случаях, когда не указана опция --mmap-pages.

--kernel-syscall-graph
Показывать цепочки вызовов ядра на пути выхода syscall.

--max-events=N
Завершать работу после обработки N событий. Отметим, что события типа strace учитываются только на выходе или при прерывании syscall, т. е. в таких случаях эта опция задаёт число выводимых строк.

--max-stack
Задаёт глубину стека при анализе цепочек вызовов. Выходящие за пределы заданной глубины стека вызовы не учитываются при анализе. Отметим, что опция влияет лишь на представление, т. е. работа ядра не ограничивается заданным параметром. Издержки, связанные с цепочкой вызовов можно установить с помощью --call-graph dwarf.
Подразумевается опция --call-graph dwarf при отсутствии --call-graph в командной строке систем с поддержкой анализа DWARF. По умолчанию используется значение из файла /proc/sys/kernel/perf_event_max_stack (при его наличии) для измерений «вживую» (без опции —input/-i) и 127 в остальных случаях.

--min-stack
Устанавливает минимальную глубину стека для анализа цепочек вызовов. Цепочки с меньшей глубиной игнорируются. По умолчанию ограничение не задано.
Подразумевается опция --call-graph dwarf при отсутствии --call-graph в командной строке систем с поддержкой анализа DWARF.

--print-sample
Выводится информация PERF_RECORD_SAMPLE и PERF_SAMPLE_ для точек трассировки raw_syscalls:sys_{enter,exit} с отладочными целями.

--proc-map-timeout
Обработка имеющихся потоков /proc/XXX/mmap может занимать много времени и эта опция позволяет установить тайм-аут (по умолчанию 500 мсек).

--sort-events
Выполнять сортировку по «пакетам» событий. Используется при обнаружении неупорядоченных событий, например при переносе потока на другой CPU в процессе обработки syscall.

--map-dump
Дамп отображений BPF по события, заданным опцией -e, например, augmented_raw_syscalls в tools/perf/examples/bpf/augmented_raw_syscalls.c. В настоящее время это лишь дамп логических значений отображения и целочисленных ключей, но со временем по умолчанию будут выводиться шестнадцатеричные значения с использованием BTF (при доступности), а также будет использоваться функция удобочитаемого вывода с использованием аргументов perf trace для сопоставления целочисленных значений со строками (pid - команда, идентификатор - имя syscall name и т. п.).

Отказы страниц

При трассировке отказов страниц используется приведённых ниже формат

```
<min|maj>fault [<ip.symbol>+<ip.offset>] ⇒ <addr.dso@addr.offset[1]> (<map type><addr level>).
```

min/maj указывает второстепенный (minor) или важный (major) отказ;

ip.symbol показывает символ для указателя инструкции (код, приведший к отказу); при отсутствии отладочных символов perf trace будет выводить raw IP[1];

addr.dso показывает объект DSO для отказавшего адреса;

map type имеет значение d (неисполняемые отображения) или x (исполняемые);

addr level имеет значение k (ядро) или dso для пользовательских объектов DSO.

Для преобразования символов может потребоваться установка отладочных файлов.

Следует принимать во внимание, что продолжительность всегда указывается нулевой и не отражает реальное время обработки отказа.

При указании опции --verbose команда perf trace пытается выводить всю информацию для IP и адреса отказа в форме dso@symbol[2]+offset.

Примеры

Трассировка только важных отказов страниц

```
$ perf trace --no-syscalls -F
```

Трассировка syscall, важных и второстепенных отказов страниц

```
$ perf trace -F all
```

```
1416.547 ( 0.000 ms): python/20235 majfault [CRYPTO_push_info_+0x0] => /lib/x86_64-linux-gnu/libcrypto.so.1.0.0@0x61be0 (x.)
```

Можно видеть важный отказ в процессе python от CRYPTO_push_info_ routine, произошедший в libcrypto.so.

[1]Instruction Pointer - указатель на инструкцию.

Трассировка первых 4 системных вызовов open, openat или open_by_handle_at (в будущем могут появиться другие вызовы open*):

```
$ perf trace -e open* --max-events 4
[root@jouet perf]# trace -e open* --max-events 4
2272.992 ( 0.037 ms): gnome-shell/1370 openat(dfd: CWD, filename: /proc/self/stat) = 31
2277.481 ( 0.139 ms): gnome-shell/3039 openat(dfd: CWD, filename: /proc/self/stat) = 65
3026.398 ( 0.076 ms): gnome-shell/3039 openat(dfd: CWD, filename: /proc/self/stat) = 65
4294.665 ( 0.015 ms): sed/15879 openat(dfd: CWD, filename: /etc/ld.so.cache, flags: CLOEXEC) = 3
```

Трассировка первого второстепенного отказа страницы при работе задачи

```
# perf trace -F min --max-stack=7 --max-events 1 sleep 1
   0.000 ( 0.000 ms): sleep/18006 minfault [__clear_user+0x1a] => 0x5626efa56080 (?k)
                                      __clear_user ([kernel.kallsyms])
                                      load_elf_binary ([kernel.kallsyms])
                                      search_binary_handler ([kernel.kallsyms])
                                      __do_execve_file.isra.33 ([kernel.kallsyms])
                                      __x64_sys_execve ([kernel.kallsyms])
                                      do_syscall_64 ([kernel.kallsyms])
                                      entry_SYSCALL_64 ([kernel.kallsyms])
```

Трассировка следующего второстепенного отказа страницы, который произошёл на первом CPU

```
# perf trace -F min --call-graph=dwarf --max-events 1 --cpu 0
    0.000 ( 0.000 ms): Web Content/17136 minfault [js::gc::Chunk::fetchNextDecommittedArena+0x4b] =>
0x7fbe6181b000 (?.)
                               js::gc::FreeSpan::initAsEmpty (inlined)
                               js::gc::Arena::setAsNotAllocated (inlined)
                               js::gc::Chunk::fetchNextDecommittedArena (/usr/lib64/firefox/libxul.so)
                               js::gc::Chunk::allocateArena (/usr/lib64/firefox/libxul.so)
                               js::gc::GCRuntime::allocateArena (/usr/lib64/firefox/libxul.so)
                               js::gc::ArenaLists::allocateFromArena (/usr/lib64/firefox/libxul.so)
                               js::gc::GCRuntime::tryNewTenuredThing<JSString, (js::AllowGC)1> (inlined)
                               js::AllocateString<JSString, (js::AllowGC)1> (/usr/lib64/firefox/libxul.so)
                               js::Allocate<JSThinInlineString, (js::AllowGC)1> (inlined)
                               JSThinInlineString::new_<(js::AllowGC)1> (inlined)
                               AllocateInlineString<(js::AllowGC)1, unsigned char> (inlined)
                               js::ConcatStrings<(js::AllowGC)1> (/usr/lib64/firefox/libxul.so)
                               [0x18b26e6bc2bd] (/tmp/perf-17136.map)
```

Трассировка двух следующих событий sched:sched_switch, четырёх событий block:*_plug, следующего block:*_unplug и следующих трёх событий net:*dev_queue с глубиной стека не более 16

```
#  perf  trace  -e  sched:*switch/nr=2/,block:*_plug/nr=4/,block:*_unplug/nr=1/,net:*dev_queue/nr=3,max-
stack=16/
             0.000 :0/0 sched:sched_switch:swapper/2:0 [120] S ==> rcu_sched:10 [120]
             0.015 rcu_sched/10 sched:sched_switch:rcu_sched:10 [120] R ==> swapper/2:0 [120]
           254.198 irq/50-iwlwifi/680 net:net_dev_queue:dev=wlp3s0 skbaddr=0xffff93498051f600 len=66
                                        __dev_queue_xmit ([kernel.kallsyms])
           273.977 :0/0 net:net_dev_queue:dev=wlp3s0 skbaddr=0xffff93498051f600 len=78
                                        __dev_queue_xmit ([kernel.kallsyms])
           274.007 :0/0 net:net_dev_queue:dev=wlp3s0 skbaddr=0xffff93498051ff00 len=78
                                        __dev_queue_xmit ([kernel.kallsyms])
          2930.140 kworker/u16:58/2722 block:block_plug:[kworker/u16:58]
          2930.162 kworker/u16:58/2722 block:block_unplug:[kworker/u16:58] 1
          4466.094 jbd2/dm-2-8/748 block:block_plug:[jbd2/dm-2-8]
          8050.123 kworker/u16:30/2694 block:block_plug:[kworker/u16:30]
          8050.271 kworker/u16:30/2694 block:block_plug:[kworker/u16:30]
```

script

Читает входной файл perf.data (созданный perf record) и выводит записанную трассировку. Существует несколько вариантов использования perf script, описанных ниже.

perf script

Для просмотра детальной трассировки записанной задачи. Можно также запустить набор подготовленных заранее сценариев, затем объединить и обобщить необработанные данные трассировки различными способами (список сценариев можно посмотреть с помощью команды perf script -l). Приведённые ниже варианты позволяют записать и запустить эти сценарии.

perf script record <script> <command>

Записывает события, требуемые для perf script report. Параметр <script> указывает имя из списка perf script --list, т. е. имя реального сценария без языкового расширения (например .pl). Если параметр <command> не задан, события записываются с использованием perf record -a.

perf script report <script> [args]

Для запуска сценария <script> и вывода результатов трассировки. Параметр <script> указывает имя из списка perf script --list, т. е. имя реального сценария без языкового расширения (например .pl). Используется вывод perf.data из предыдущего запуска perf script record <script>, который следует сохранить для работы этой команды. Параметр [args] указывает аргументы (в основном необязательные), которых ждёт сценарий.

perf script <script> <required-script-args> <command>

Запись событий, нужных для <script> и запуск <script> в режиме live-mode, т. е. без записи данных на диск. Параметр <script> указывает имя из списка perf script --list, т. е. имя реального сценария без языкового расширения (например .pl). Если параметр <command> не задан, события записываются с использованием perf record -a. Если для <script> нужны аргументы, из следует указать до <command>. В этом режиме не поддерживаются необязательные аргументы и если такие аргументы нужны, их можно задать в отдельных командах perf script record и perf script report с выводом результатов записи в stdout, конвейером в stdin для сценария report и опциями -o - и -i - в соответствующих командах.

perf script <top-script>

Запись событий, нужных для <top-script> и запуск <top-script> в режиме live-mode без записи промежуточных результатов на диск. Параметр <top-script> указывает имя из списка perf script --list, т. е. имя реального сценария без языкового расширения (например .pl), которое может быть именем любого сценария с суффиксом top.

Параметр [<record-options>] может передаваться на этап записи perf script record и в варианты live-mode, но не поддерживается для вариантов <top-script> live-mode и perf script report.

Дополнительная информация о сценариях приведена в разделах script-perl и script-python .

Синтаксис

```
perf script [<options>]
perf script [<options>] record <script> [<record-options>] <command>
perf script [<options>] report <script> [script-args]
perf script [<options>] <script> <required-script-args> [<record-options>] <command>
perf script [<options>] <top-script> [script-args]
```

Опции

<command>...
Любая команда, разрешённая командным процессором.

-D, --dump-raw-trace=
Вывод подробного дампа данных трассировки.

-L, --Latency=
Вывод атрибутов задержки (запрет irqs/preemption и т. п.).

-l, --list=
Вывод списка доступных сценариев трассировки.

-s [lang], --script=
Обработка данных трассировки указанным сценарием ([lang]:script[.ext]). Если параметр указан вместо имени сценария, выводится список доступных языков сценариев.

-g, --gen-script=
Генерировать стартовый сценарий perf-script.[ext] для данного языка с использованием текущего файла perf.data.

-a
Форсирует сбор данных в масштабе всей системы. Сценарии, запускаемые без параметра <command>, обычно используют опцию -a по умолчанию, а при использовании с <command> эта опция может быть указана отдельно, если нужен сбор данных от системы в целом.

-i, --input=
Задаёт имя входного файла (по умолчанию perf.data, если stdin не является fifo).

-d, --debug-mode
Выполнять различные проверки, такие как порядок выборок и потерянные события.

-F, --fields
Список полей вывода через запятые - comm, tid, pid, time, cpu, event, trace, ip, sym, dso, addr, symoff, srcline, period, iregs, uregs, brstack, brstacksym, flags, bpf-output, brstackinsn, brstackoff, callindent, insn, insnlen, synth, phys_addr, metric, misc, srccode. Перед списком полей указывается тип трассировки (trace, sw или hw), для индикации типа событий, к которым относится список полей, например -F sw:comm,tid,time,ip,sym или -F trace:time,cpu,trace. Команда

```
perf script -F <fields>
```

эквивалентна

```
perf script -F trace:<fields> -F sw:<fields> -F hw:<fields>
```

т. е. указанные поля относятся ко всем типам событий, если конкретный тип не указан.
В дополнение к переопределению полей можно добавлять и удалять принятые по умолчанию поля. Например,

```
-F -cpu,+insn
```

удаляет поле cpu и добавляет поле insn. Добавление и удаление полей нельзя смешивать с обычным переопределением.
Аргументы обрабатываются в порядке их следования и последующее указание может сбросить предшествующее. Например,

```
-F trace: -F comm,tid,time,ip,sym
```

Первая опция -F подавляет события трассировки (список полей пуст - ""), а втора устанавливает для всех типов поля comm,tid,time,ip,sym. В таких случаях выдаются предупреждения вида

```
"Overriding previous field request for all events."
```

В другом наборе

```
-F comm,tid,time,ip,sym -F trace:
```

первая опция -F задаёт поля для всех событий, а вторая подавляет события трассировки. Пользователь получит сообщение о переопределении и в результате для указанных полей будут выводиться лишь события SW и HW.
Можно добавить и удалить поля лишь для определённого типа. Например,

```
-Fsw:-cpu,-period
```

удалит поля cpu и period для программных событий.
Если при использовании «шаблонной» опции указать поле, которое не недействительно для того или иного типа событий, будет выдано сообщение об игнорировании поля для этого типа. Например,

```
$ perf script -F comm,tid,trace
'trace' not valid for hardware events. Ignoring.
'trace' not valid for software events. Ignoring.
```

Если же указано недействительное поле для конкретного типа, возникает ошибка. Например,

```
perf script -v -F sw:comm,tid,trace
'trace' not valid for software events.
```

В этом случае perf script завершает работу по ошибке.
Поле флагов синтезируется и может иметь значение при декодировании Instruction Trace. Флаги bcrosyiABEx указывают ветвление (branch), вызов (call), возврат (return), условие (conditional), системное (system) или асинхронное (asynchronous) событие, прерывание (interrupt), прерывание транзакции (transaction abort), начало (trace begin) и завершение (trace end) трассировки и нахождение в транзакции (in transaction), соответственно. Известные комбинации флагов указываются более изящно, например, call для bc, return для br, jcc для bo, jmp для b, int для bci, iret для bri, syscall для bcs, sysret для brs, async для by, hw int для bcyi, tx abrt для bA, tr strt для bB, tr end для bE. Однако флаг x в таких случаях будет выводиться отдельно, например, jcc (x) для условного ветвления внутри транзакции.
Поле callindent синтезируется и может иметь значение при декодировании Instruction Trace. Для вызовов и возвратов оно будет указывать имя символа, окружённое пробелами для отражения глубины стека.
При декодировании трассировки инструкций insn и insnlen дают байты и размер для текущей инструкции.
Поле synth используется синтезированными событиями, которые могут создаваться при декодировании Instruction Trace.
Пользователь не может задавать отсутствие полей для всех событий, т. е. -F "" недопустимо.

Вывод brstack включает связанную с ветвлениями информацию с необработанными адресами, используя синтаксис /v/v/v/v/cycles, в следующем порядке:

FROM инструкция источника ветвления;
TO инструкция цели ветвления;
M/P/- M - цель или направление ветвления предсказано не верно, P - цель или направление предсказаны, - - не поддерживается;
X/- X - ветвление внутри области транзакции, - - ветвление вне транзакции или не поддерживается;
A/- A - запись прерывания TSX, - - не прерываемая область или не поддерживаемые циклы.

Поле brstacksym похоже на brstack, но адреса FROM и TO выводятся в символьной форме, если это возможно.

При задании brstackinsn выводятся полные ассемблерные последовательности ветвлений для каждой выборки (полный путь исполнения, ведущий к выборке). Для поддержки такой возможности запись данных должна быть выполнена командой perf record с опцией -b или -j any.

Поле brstackoff указывает смещение в конкретный объект dso или двоичный файл.

С опцией metric команда perf script может рассчитывать параметры для периодов выборки подобно perf stat. Для этого нужно указать группу, имеющую множество метрик, с опцией :S для perf record. Тогда perf будет выбирать первое событие и рассчитывать метрики для всех событий в группе. Важно отметить, что рассчитанные параметры дают среднее значение за период выборки а не мгновенное в точке выборки.

Для событий выборки можно выводить поле misc, указав опцию field with F +misc с последующими символами для отображаемых битов, как показано ниже.

```
PERF_RECORD_MISC_KERNEL               K
PERF_RECORD_MISC_USER                 U
PERF_RECORD_MISC_HYPERVISOR           H
PERF_RECORD_MISC_GUEST_KERNEL         G
PERF_RECORD_MISC_GUEST_USER           g
PERF_RECORD_MISC_MMAP_DATA*           M
PERF_RECORD_MISC_COMM_EXEC            E
PERF_RECORD_MISC_SWITCH_OUT           S
PERF_RECORD_MISC_SWITCH_OUT_PREEMPT   Sp

$ perf script -F +misc ...
 sched-messaging  1414 K     28690.636582:       4590 cycles ...
 sched-messaging  1407 U     28690.636600:     325620 cycles ...
 sched-messaging  1414 K     28690.636608:      19473 cycles ...
 поле misc  __________/
```

-k, --vmlinux=<file>
Путь к файлу vmlinux.

--kallsyms=<file>
Путь к файлу kallsyms.

--symfs=<directory>
Каталог, от которого начинается поиск файлов с символами.

-G, --hide-call-graph
Отключает вывод цепочек вызовов при отображении символов.

--stop-bt
Выключает отображение графа вызовов по этим символам.

-C, --cpu
Задаёт включение в отчёт лишь выборок для указанных списком CPU. Процессоры можно перечислить через запятые (0,1) или указать диапазон (0-2) без пробелов. По умолчанию учитываются все CPU.

-c, --comms=
Задаёт учёт символов лишь в указанных столбцах. Можно использовать файл CSV, заданный параметром file://filename.

--pid=
Задаёт вывод событий лишь для указанного идентификатора процесса (список разделённых запятыми значений).

--tid=
Задаёт вывод событий лишь для указанного идентификатора потока (список разделённых запятыми значений).

-I, --show-info
Задаёт вывод расширенной информации о файле perf.data. Эти данные могут быть очень велики и занять много места на экране. В настоящее время они включают топологию cpu и numa для хост-системы. Опцию можно использовать лишь в режиме perf script report.

--show-kernel-path
Пытаться преобразовать (resolve) путь [kernel.kallsyms].

--show-task-events
Отображать связанные с задачей события (например, FORK, COMM, EXIT).

--show-mmap-events
Отображать связанные с mmap события (например, MMAP, MMAP2).

--show-namespace-events
Отображать события пространства имён (PERF_RECORD_NAMESPACES).

--show-switch-events
Отображать события переключения контекста (PERF_RECORD_SWITCH и PERF_RECORD_SWITCH_CPU_WIDE).

--show-lost-events
Отображать события потерь (PERF_RECORD_LOST).

--show-round-events
Отображать события законченного цикла (finished round) (PERF_RECORD_FINISHED_ROUND).

--demangle
Восстанавливать имена символов в удобочитаемую форму. По умолчанию включено, отключается --no-demangle.

--demangle-kernel
Восстанавливать имена символов ядра в удобочитаемую форму (для ядер C++).

--header
Показывать заголовок perf.data.

--header-only
Показывать только заголовок perf.data.

--itrace

Задаёт опции декодирования данных трассировки инструкций:

i синтезировать события инструкций;
b синтезировать события ветвления;
c синтезировать события ветвления (только вызовы);
r синтезировать события ветвления (только возвраты);
x синтезировать события транзакций;
w синтезировать события ptwrite;
p синтезировать события, связанные с питанием;
e синтезировать события, связанные с ошибками;
d создать журнал отладки;
g синтезировать цепочку вызовов (используется с i или x);
l синтезировать записи последнего ветвления (используется с i или x);
s пропустить указанное число записей в начале.

По умолчанию учитываются все события (--itrace=ibxwpe), за исключением perf (--itrace=ce)

В дополнение к этому можно задать интервал учёта (по умолчанию 100000, за исключением perf script, где устанавливается 1) в удобных единицах:

i инструкции;
t такты часов;
ms миллисекунды;
us микросекунды;
ns наносекунды (используется по умолчанию).

Можно также указать размер цепочки (по умолчанию 16, максимум 1024) для событий.

Можно указать также число записей последнего ветвления (по умолчанию 64, максимум 1024).

Можно пропустить генерацию событий (инструкции, ветвления, транзакции, ptwrite, питание) в начале. Это удобно для пропуска фазы инициализации. Например, опция

```
--itrace=i0nss1000000
```

обеспечит пропуск первого миллиона инструкций.

Для полного запрета декодирования служит опция --no-itrace.

--full-source-path

Выводит полный путь к файлам исходного кода при выводе номеров строк.

--max-stack

Задаёт ограничение глубины стека при разборе цепочки вызовов. Это задаёт компромисс между объёмом информации и скоростью обработки для заданий, имеющих очень большую глубину стека вызовов. Отметим, что при использовании опции --itrace синтезированный размер цепочки вызовов будет переопределять это значение, если оно меньше синтезированного. По умолчанию используется глубина стека 127.

--ns

Задаёт использование 9 десятичных цифр при отображении времени (т. е. показывать наносекунды).

-f, --force

Не проверять принадлежность.

--time

Задаёт анализ выборок лишь из временного окна <start>,<stop>. Время указывается в формате секунды.микросекунды. Если время начала не указано (т. е. ,x.y), анализ начинается с начала файла. Если время окончания не указано (т. е. x.y,), анализ выполняется до конца файла.

Поддерживается также указание интервалов в процентах - a%/n,b%/m,... или a%-b%,c%-%d,...

Например, для выбора вторых 10% интервала записи можно использовать команду

```
perf script --time 10%/2
```

Для выбора интервала от 0% до 10% служит команда

```
perf script --time 0%-10%
```

Например, для выбора первых и вторых 10% интервала записи можно использовать команду

```
perf script --time 10%/1,10%/2
```

Для выбора интервалов от 0% до 10% и от 30% до 40% служит команда

```
perf script --time 0%-10%,30%-40%
```

--max-blocks

задаёт максимальное число программных блоков при выводе с brstackasm для каждой выборки.

--reltime

Задаёт вывод временных меток от начала трассировки.

--per-event-dump

Создаёт для событий файлы perf.data.EVENT.dump вместо вывода на stdout (полезно, например, для создания графа кадров).

--inline

Если адрес графа вызовов относится к inline-функции, будет выводиться inline-стек, каждая запись которого указывает имя функции или файл и строку. Опция включена по умолчанию, отключается с помощью -no-inline.

--insn-trace

Показывать поток инструкций для трассировки intel_pt. При использовании с --xed будет выполнено реассемблирование.

--xed

Запустить на выходе дизассемблер xed (должен быть установлен).

--call-trace

Показывать поток вызовов для трассировок intel_pt. Процессоры будут чередоваться, но могут быть отфильтрованы опцией -C.

--call-ret-trace

Показывать поток вызовов и возвратов для трассировок intel_pt.

--graph-function

Для itrace задаёт вывод только указанных функций и вызываемых ими для itrace. Функции указываются через запятые.

script-perl

Эта команда служит для обработки данных трассировки с помощью сценариев Perl и встроенного в perf интерпретатора Perl. Данные считываются из входного файла и обрабатываются указанным сценарием Perl.

Синтаксис

```
perf script [-s [Perl]:script[.pl] ]
```

Стартовые сценарии

Запуск команды perf script -g perl из каталога с файлом perf.data приведёт к созданию стартового сценария, содержащего обработчик для каждого типа событий в файле трассировки. Будут выведены доступные поля для каждого события в файле трассировки.

Можно также посмотреть типовые примеры в каталоге /usr/libexec/perf-core/scripts/perl. Сценарий check-perf-script.pl, хотя и не даёт интересных результатов, пытается выполнить основные функции сценариев.

Обработчики событий

При вызове perf script с использованием сценария трассировки, вызывается заданный пользователем обработчик функций для каждого события трассировки. Если для данного типа события обработчик не определён, событие игнорируется (или передаётся функции trace_unhandled, описанной ниже).

Большинство значений полей события передаётся обработчику в качестве аргументов, но для части полей снова вызывается исполняемый файл perf, как описано ниже.

Например, приведённая ниже команда perf record может использоваться для записи всех событий sched_wakeup в системе.

```
# perf record -a -e sched:sched_wakeup
```

Трассировки, предназначенные для обработки сценарием, следует записывать с опцией -a (система в целом).

Файл /sys/kernel/debug/tracing/events/sched/sched_wakeup/format для событий sched_wakep определяет перечисленные ниже поля.

```
format:
        field:unsigned short common_type;
        field:unsigned char common_flags;
        field:unsigned char common_preempt_count;
        field:int common_pid;

        field:char comm[TASK_COMM_LEN];
        field:pid_t pid;
        field:int prio;
        field:int success;
        field:int target_cpu;
```

Обработчик для этого события определён ниже.

```
sub sched::sched_wakeup
{
   my ($event_name, $context, $common_cpu, $common_secs,
       $common_nsecs, $common_pid, $common_comm,
       $comm, $pid, $prio, $success, $target_cpu) = @_;
}
```

Функция обработки принимает форму subsystem::event_name.

Аргументы $common_* в списке аргументов обработчика - это аргументы, передаваемые всем обработчикам событий. Некоторые из полей соответствуют полям common_* в файле format, часть полей синтезируется, а часть полей common_* не передаётся каждому обработчику событий в качестве аргументов, но они доступны как библиотечные функции.

Ниже приведено краткое описание инвариантных аргументов событий.

$event_name	имя события в текстовой форме.
$context	не обрабатываемое значение cookie, используемое при обратных вызовах perf.
$common_cpu	процессор, где произошло событие.
$common_secs	секунды из временной метки события.
$common_nsecs	наносекунды из временной метки события.
$common_pid	pid текущей задачи.
$common_comm	имя текущего процесса.

Все остальные поля в файле формата события имеют дубликаты в виде одноимённых аргументов функций, как можно видеть из приведённого выше примера.

Выше представлена база для прямого доступа к каждому полю каждого события и это покрывает 90% того, что нужно знать для создания сценариев трассировки. Остальное описано ниже.

Схема сценария

Каждый Perl-сценарий perf script следует начинать с указания пути поиска модулей Perl и операторов use для нескольких модулей поддержки.

```
use lib "$ENV{'PERF_EXEC_PATH'}/scripts/perl/Perf-Trace-Util/lib";
use lib "./Perf-Trace-Util/lib";
use Perf::Trace::Core;
use Perf::Trace::Context;
use Perf::Trace::Util;
```

Остальная часть сценария может включать функции обработки и поддержки в любом порядке. Кроме упомянутых выше функций сценарий может включать необязательные функции, перечисленные ниже.

trace_begin

При наличии этой функции она вызывается перед обработкой каждого события и позволяет сценарию задать нужные параметры.

```
sub trace_begin
{
}
```

trace_end

При наличии этой функции она вызывается после обработки всех событий и позволяет сценарию выполнить финальные задачи, такие как вывод результатов.

```
sub trace_end
{
}
```

trace_unhandled

При наличии этой функции она вызывается для всех событий, не имеющих явных обработчиков. Функции передаётся стандартный набор базовых аргументов.

```
sub trace_unhandled
{
    my ($event_name, $context, $common_cpu, $common_secs,
        $common_nsecs, $common_pid, $common_comm) = @_;
}
```

Далее описываются доступные модули встроенного интерпретатора Perl и связанные с ними функции.

Доступные модули и функции

Здесь описаны функции и переменные, доступные в модулях Perf::Trace::*. Для их использования из данного сценария в него добавляются соответствующие строки use Perf::Trace::XXX.

Perf::Trace::Core

Этот модуль включает некоторые важные для пользовательских сценариев функции.

Функции flag_str и symbol_str представляют удобочитаемые строки для флагов и символьных полей, соответствующие строкам, извлекаемым ил полей print fmt форматных файлов событий:

flag_str($event_name, $field_name, $field_value) возвращает строку, соответствующую $field_value для поля флагов $field_name в $event_name;

symbol_str($event_name, $field_name, $field_value) возвращает строку, соответствующую $field_value для символьного поля $field_name в $event_name.

Perf::Trace::Context

Некоторые из полей в форматном файле события не являются общими, но нужны для сценария.

Модуль Perf::Trace::Context определяет набор функций, которые могут служить для доступа к этим данным в контексте текущего события. Каждая из этих функций ждёт переменную $context, которая совпадает с одноименной переменной, передаваемой каждому обработчику событий во втором аргументе.

common_pc($context) возвращает счётчик common_preempt для текущего события;

common_flags($context) возвращает поле common_flags для текущего события;

common_lock_depth($context) возвращает common_lock_depth для текущего события.

Perf::Trace::Util

Различные вспомогательные функции для perf script.

nsecs($secs, $nsecs) возвращает число наносекунд для данной пары «секунды-наносекунды»;

nsecs_secs($nsecs) возвращает число целых секунд из значения в наносекундах;

nsecs_nsecs($nsecs) возвращает число наносекунд, оставшихся после извлечения целых секунд;

nsecs_str($nsecs) возвращает печатаемую строку в форме секунды.наносекунды;

avg($total, $n) возвращает среднее значение на основе суммы и числа значений.

script-python

Обработка данных трассировки в сценарии Python с использованием встроенного в perf интерпретатора Python. Выполняется считывание и обработка входного файла с выводом результатов обработки данных трассировки сценарием Python.

Синтаксис

```
perf script [-s [Python]:script[.py] ]
```

Короткий пример

Здесь показан процесс создания рабочего сценария Python, который агрегирует и извлекает полезную информацию из необработанного потока perf script.

В приведённом ниже примере рассмотрены этапы создания сценария syscall-counts, который можно увидеть в списке, выводимом по команде perf script -l. Сценарий также показывает способ интеграции в список сценариев общего пользования perf script, выводимых по этой команде.

Сценарий syscall-counts очень просто, но демонстрирует основные идеи, требуемые для создания полезных сценариев. Ниже представлен пример вывода этого сценария (вместо имён вызовов syscall, которые ещё не поддерживаются, указаны номера).

```
syscall events:

event                                          count
----------------------------------------  -----------
sys_write                                     455067
sys_getdents                                    4072
sys_close                                       3037
sys_swapoff                                     1769
sys_read                                         923
sys_sched_setparam                               826
sys_open                                         331
sys_newfstat                                     326
sys_mmap                                         217
sys_munmap                                       216
sys_futex                                        141
sys_select                                       102
sys_poll                                          84
sys_setitimer                                     12
sys_writev                                         8
15                                                 8
sys_lseek                                          7
sys_rt_sigprocmask                                 6
sys_wait4                                          3
sys_ioctl                                          3
sys_set_robust_list                                1
sys_exit                                           1
56                                                 1
sys_access                                         1
```

Задача по сути состоит в поддержке значения для каждого syscall, обновляемых при каждом вызове в системе. Сценарий делает это, но сначала нужно записать данные, которые будут обрабатываться этим сценарием. Теоретически есть несколько способов решения задачи.

- Можно включить каждое событие в каталоге tracing/events/syscalls, но там более 600 syscall, что выходит за пределы возможностей perf. Однако эти отдельные события syscall будут полезны впредь для детализации отдельных syscall.
- Можно включить sys_enter и/или sys_exit в tracing/events/raw_syscalls. Они вызываются для всех syscall и поле id можно использовать для идентификации отдельных syscall.

Для этого сценария нужно знать лишь о входе в syscall, поэтому будет использоваться perf record для записи лишь событий sys_enter.

```
# perf record -a -e raw_syscalls:sys_enter

^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 56.545 MB perf.data (~2470503 samples) ]
```

Опции говорят о необходимости собирать данные для каждого события syscall в системе и мультиплексировать вывод от отдельных процессоров в один поток. Этот поток будет записываться в файл perf.data в текущем каталоге.

Имея файл perf.data с нужными данными, можно использовать команду perf script -g для генерации сценария Python, который будет включать обработчик обратных вызовов для каждого типа событий в потоке данных трассировки из perf.data (см. Стартовые сценарии).

```
# perf script -g python
generated Python script: perf-script.py
```

Выходной файл также создаётся в текущем каталоге и называется perf-script.py. Этот файл приведён ниже.

```
# perf script event handlers, generated by perf script -g python
# Licensed under the terms of the GNU GPL License version 2

# The common_* event handler fields are the most useful fields common to
# all events.  They don't necessarily correspond to the 'common_*' fields
# in the format files.  Those fields not available as handler params can
# be retrieved using Python functions of the form common_*(context).
# See the perf-script-python Documentation for the list of available functions.

import os
import sys

sys.path.append(os.environ['PERF_EXEC_PATH'] + '/scripts/python/Perf-Trace-Util/lib/Perf/Trace')

from perf_trace_context import *
from Core import *

def trace_begin():
        print "in trace_begin"

def trace_end():
        print "in trace_end"

def raw_syscalls__sys_enter(event_name, context, common_cpu,
        common_secs, common_nsecs, common_pid, common_comm,
        id, args):
                print_header(event_name, common_cpu, common_secs, common_nsecs, common_pid, common_comm)

          print "id=%d, args=%s\n" % (id, args),

def trace_unhandled(event_name, context, event_fields_dict):
        print ' '.join(['%s=%s'%(k,str(v))for k,v in sorted(event_fields_dict.items())])

def print_header(event_name, cpu, secs, nsecs, pid, comm):
        print "%-20s %5u %05u.%09u %8u %-20s " % (event_name, cpu, secs, nsecs, pid, comm),
```

Файл начинается с блока комментариев, за которыми следуют операторы импорта и добавление пути, который следует включать в каждый сценарий perf script.

далее следуют сгенерированные функции trace_begin() и trace_end(), которые вызываются в начале и в конце сценария (см. Схема сценария).

Затем приведены функции обработчиков для всех событий из выходного файла perf record. Функции обработки имеют форму subsystemevent_name и именованные параметры, по одному для каждого поля события. В примере используется лишь одно событие - raw_syscallssys_enter(). Обработчики более подробно описаны ниже.

Финальная пара функций, подобно начальной и завершающей функциям, генерируется для каждого сценария. Функция trace_unhandled() вызывается при обнаружении сценарием в файле perf.data функции, для которой нет конкретного обработчика. Это может быть обусловлено наличием в файле события, которое не представляет интереса или запуском сценария, который не соответствует файлу трассировки.

Сценарий, созданный опцией -g, просто выводит строку для каждого события из потока трассировки, указывающую событие и значения его полей, на stdout. Функция print_header() выводит заголовок.

Затем можно переименовать сценарий и воспользоваться им.

```
# mv perf-script.py syscall-counts.py
# perf script -s syscall-counts.py

raw_syscalls__sys_enter     1 00840.847582083     7506 perf                 id=1, args=
raw_syscalls__sys_enter     1 00840.847595764     7506 perf                 id=1, args=
raw_syscalls__sys_enter     1 00840.847620860     7506 perf                 id=1, args=
raw_syscalls__sys_enter     1 00840.847710478     6533 npviewer.bin         id=78, args=
raw_syscalls__sys_enter     1 00840.847719204     6533 npviewer.bin         id=142, args=
raw_syscalls__sys_enter     1 00840.847755445     6533 npviewer.bin         id=3, args=
raw_syscalls__sys_enter     1 00840.847775601     6533 npviewer.bin         id=3, args=
raw_syscalls__sys_enter     1 00840.847781820     6533 npviewer.bin         id=3, args=
          .
          .
          .
```

Сценарии, конечно, нужны не просто для вывода строк событий, но этот собирает данные, которые можно использовать. Поэтому исключим из него печать, а также функции trace_begin() и trace_unhandled(), которые не нужны.

```
          import os
          import sys

          sys.path.append(os.environ['PERF_EXEC_PATH'] + \
                  '/scripts/python/Perf-Trace-Util/lib/Perf/Trace')

          from perf_trace_context import *
          from Core import *

          def trace_end():
                  print "in trace_end"

          def raw_syscalls__sys_enter(event_name, context, common_cpu,
                  common_secs, common_nsecs, common_pid, common_comm,
                  id, args):
```

В trace_end() будем просто выводить результаты, но сначала их нужно создать. Для этого нужно в обработчик событий sys_enter() включить нужные расчёты и выполнить их для каждого события. Хэш-таблица, индексируемая по идентификаторам syscall является подходящим способом сохранения информации. При каждом вызове sys_enter() инкрементируется счётчик, связанный с хэш-записью, индексированной идентификатором syscall.

```
            syscalls = autodict()

            try:
              syscalls[id] += 1
            except TypeError:
              syscalls[id] = 1
```

Объект autodict является специальным словарём Python (реализован в Core.py), поддерживающим автоматически обновляемые хэш-значения Perl в языке Python. Это позволяет назначать вложенные хэш-значения без необходимости создавать промежуточные уровни хэширования при их отсутствии. Например, syscalls[comm][pid][id] = 1 будет создавать промежуточные уровни хэширования и в конце концов присваивать значение 1 хэш-записи для идентификатора (поскольку присваиваемое значение само не является хэш-объектом, начальное значение присваивается в TypeError).

Приведённый выше код помещается в обработчик raw_syscalls__sys_enter() и это фактически даёт одноуровневый словарь с идентификатором syscall в качестве ключа и счётчиками.

Функция print_syscall_totals() перебирает записи в словаре и выводит для каждой записи строку с именем syscall (записи содержат идентификаторы syscall, которые передаются функции syscall_name(), преобразующей их в имена). Вывод выполняется после обработки всех событий в файле трассировки путём вызова print_syscall_totals() из обработчика trace_end() в конце сценария.

Окончательный сценарий приведён ниже (функция syscall_name() ещё не реализована, поэтому выводятся идентификаторы, а не имена).

```
import os
import sys

sys.path.append(os.environ['PERF_EXEC_PATH'] + '/scripts/python/Perf-Trace-Util/lib/Perf/Trace')

from perf_trace_context import *
from Core import *
from Util import *

syscalls = autodict()

def trace_end():
        print_syscall_totals()

def raw_syscalls__sys_enter(event_name, context, common_cpu,
        common_secs, common_nsecs, common_pid, common_comm, id, args):
        try:
                syscalls[id] += 1
```

```
        except TypeError:
                syscalls[id] = 1

def print_syscall_totals():
     if for_comm is not None:
             print "\nsyscall events for %s:\n\n" % (for_comm),
     else:
             print "\nsyscall events:\n\n",

        print "%-40s  %10s\n" % ("event", "count"),
        print "%-40s  %10s\n" % ("----------------------------------------", "-----------"),

        for id, val in sorted(syscalls.iteritems(), key = lambda(k, v): (v, k), reverse = True):
             print "%-40s  %10d\n" % (syscall_name(id), val),
```

Сценарий можно запустить командой

```
# perf script -s syscall-counts.py
```

Показанные выше этапы создания и запуска сценариев можно обобщить на любой набор интересующих точек трассировки. Такие точки можно определить путём просмотра списка доступных событий с помощью команды perf list или файлов в /sys/kernel/debug/tracing/events/. Затем трассировка нужных событий записывается с помощью perf record, которой указаны интересующие события, создаётся базовый сценарий с помощью команды perf script -g python и код сценария редактируется для обработки и отображения интересующих данных.

Полученный в результате сценарий можно сохранить для последующего применения. Разместив файл сценария в нужном месте, вы сможете потом найти его по команде perf script -l.

```
# perf script -l
List of available trace scripts:
  wakeup-latency                   system-wide min/max/avg wakeup latency
  rw-by-file <comm>                r/w activity for a program, by file
  rw-by-pid                        system-wide r/w activity
  syscall-counts                   system-wide syscall counts
```

Стартовые сценарии

Можно быстро создать базовый сценарий для конкретного набора данных с помощью команды perf script -g python, запущенной из каталога, в котором хранится файл perf.data. Полученный в результате сценарий будет включать обработчики для каждого типа событий в файле трассировки. Эти обработчики просто выводят все поля для каждого события в одну строку.

Можно также воспользоваться сценариями из каталога /usr/libexec/perf-core/scripts/python, где представлено несколько базовых примеров.

Обработчики событий

При вызове perf script со сценарием трассировки вызывается определённая пользователем функция обработки для каждого события. Если для какого-либо событие обработчик не задан, событие игнорируется или передаётся функции trace_unhandled.

Большинство значений полей события передаётся обработчику в качестве аргументов, но для части полей снова вызывается исполняемый файл perf, как описано ниже.

Например, приведённая ниже команда perf record может использоваться для записи всех событий sched_wakeup в системе.

```
# perf record -a -e sched:sched_wakeup
```

Трассировки, предназначенные для обработки сценарием, следует записывать с опцией -a (система в целом).

Файл /sys/kernel/debug/tracing/events/sched/sched_wakeup/format для событий sched_wakep определяет перечисленные ниже поля.

```
 format:
        field:unsigned short common_type;
        field:unsigned char common_flags;
        field:unsigned char common_preempt_count;
        field:int common_pid;

        field:char comm[TASK_COMM_LEN];
        field:pid_t pid;
        field:int prio;
        field:int success;
        field:int target_cpu;
```

Обработчик для этого события определён ниже.

```
def sched__sched_wakeup(event_name, context, common_cpu, common_secs,
       common_nsecs, common_pid, common_comm,
       comm, pid, prio, success, target_cpu):
       pass
```

Функция обработки принимает форму subsystem__event_name.

Аргументы common_* в списке аргументов обработчика - это аргументы, передаваемые всем обработчикам событий. Некоторые из полей соответствуют полям common_* в файле format, часть полей синтезируется, а часть полей common_* не передаётся каждому обработчику событий в качестве аргументов, но они доступны как библиотечные функции.

Ниже приведено краткое описание инвариантных аргументов событий.

event_name	имя события в текстовой форме.
context	не обрабатываемое значение cookie, используемое при обратных вызовах perf.
common_cpu	процессор, где произошло событие.
common_secs	секунды из временной метки события.

common_nsecs наносекунды из временной метки события.

common_pid pid текущей задачи.

common_comm имя текущего процесса.

Все остальные поля в файле формата события имеют дубликаты в виде одноимённых аргументов функций, как можно видеть из приведённого выше примера.

Выше представлена база для прямого доступа к каждому полю каждого события и это покрывает 90% того, что нужно знать для создания сценариев трассировки. Остальное описано ниже.

Схема сценария

Каждый сценарий Python для perf script должен начинаться с установки пути поиска модулей Python и импорта нескольких модулей поддержки, как показано ниже

```
import os
import sys

sys.path.append(os.environ['PERF_EXEC_PATH']+'/scripts/python/Perf-Trace-Util/lib/Perf/Trace')


from perf_trace_context import *
from Core import *
```

Остальная часть сценария может содержать обработчики и функции поддержки в любом порядке. Кроме упомянутых выше обработчиков событий каждый сценарий может включать перечисленные ниже необязательные функции.

trace_begin

При наличии этой функции она вызывается перед обработкой каждого события и позволяет сценарию задать нужные параметры.

```
def trace_begin():
    pass
```

trace_end

При наличии этой функции она вызывается после обработки всех событий и позволяет сценарию выполнить финальные задачи, такие как вывод результатов.

```
def trace_end():
    pass
```

trace_unhandled

При наличии этой функции она вызывается для всех событий, не имеющих явных обработчиков. Функции передаётся стандартный набор базовых аргументов.

```
def trace_unhandled(event_name, context, event_fields_dict):
    pass
```

Далее описываются встроенные модули Python для perf script и их функции.

Доступные модули и функции

Ниже описаны функции и переменные, доступные в модулях Python для perf script. Для использования их из того или иного модуля нужно добавить в него строку import с именем импортируемого сценария.

Core.py

Модуль содержит функции, требуемые для пользовательских сценариев.

Функции flag_str и symbol_str обеспечивают удобочитаемое представление полей флагов и символьных полей. Это соответствует строкам и значениям, извлекаемым из полей print fmt в форматных файлах событий:

flag_str(event_name, field_name, field_value) возвращает строку, соответствующую field_value для поля флагов в field_name имени события event_name;
symbol_str(event_name, field_name, field_value) возвращает строку, соответствующую field_value для символьного поля field_name события event_name.

Функция autodict возвращает специальный словарь Python, реализующий автоматически обновляемые хэш-значения Perl в языке Python. Это позволяет присваивать вложенные хэш-значения без проблем, связанных с промежуточными уровнями, если они отсутствуют.

autodict() возвращает экземпляр словаря.

perf_trace_context

Некоторые из базовых полей форматного файла события мало распространены, но нужны для пользовательских сценариев. Модуль perf_trace_context определяет набор функций для доступа к таким данным в контексте текущего события. Каждая из этих функций ждёт переменную контекста, которая совпадает с передаваемой обработчикам событий во втором аргументе.

common_pc(context) возвращает счётчик common_preempt для текущего события;

common_flags(context) возвращает common_flags для текущего события;

common_lock_depth(context) возвращает common_lock_depth для текущего события.

Util.py

Вспомогательные функции для использования с perf script:

nsecs(secs, nsecs) возвращает общее число наносекунд для данной пары секунды-наносекунды;

nsecs_secs(nsecs) возвращает число целых секунд, заданных в наносекундах;

nsecs_nsecs(nsecs) возвращает число наносекунд, оставшихся после целых секунд;

nsecs_str(nsecs) возвращает печатаемую строку в форме секунды.наносекунды;

avg(total, n) возвращает среднее значение по сумме и количеству значений. number of values

Поддерживаемые поля

В настоящее время поддерживаются поля ev_name, comm, pid, tid, cpu, ip, time, period, phys_addr, addr, symbol, dso, time_enabled, time_running, values, callchain, brstack, brstacksym, datasrc, datasrc_decode, iregs, uregs, weight, transaction, raw_buf, attr.

Некоторые поля имею субэлементы:

brstack - from, to, from_dsoname, to_dsoname, mispred, predicted, in_tx, abort, cycles;

brstacksym - from, to, pred, in_tx, abort (преобразованная строка).

Например, можно использовать приведённый ниже код для вывода значений brstack "from", "to", "cycles".

```
if brstack in dict: for entry in dict[brstack]: print "from %s, to %s, cycles %s" % (entry["from"], entry["to"], entry["cycles"])
```

timechart

Инструмент для визуализации поведения системы под нагрузкой. Есть 2 варианта применения perf timechart.

perf timechart record <command>

Запись событий системного уровня при выполнении произвольной задачи. По умолчанию записываются только события планировщиков и CPU (переключение задач, время работы, состояние питания CPU и т. п.), но можно записать также события ввода-вывода (диски, сеть) с помощью опции -I.

perf timechart

Трассировка в файл SVG[1], который можно просматривать в таких программах как Inkscape. В зависимости от событий в файле perf.data режим timechart будет показывать события планировщика, процессоров и ввода-вывода. В режиме IO каждая панель включает два графика (верхний и нижний). На верхнем показаны входящие события (чтение дисков, входящие из сети пакеты), а на нижнем - исходящие (запись на диск, передача пакетов в сеть). Имеются также панели, показывающие время пребывания программы в системных вызовах poll/epoll/select.

Опции режима timechart

-o, --output=

Имя выходного файла (по умолчанию output.svg).

-i, --input=

Имя входного файла (по умолчанию perf.data, если stdin не является fifo).

-w, --width=

Выбор ширины для файла SVG (по умолчанию 1000).

-P, --power-only

Вывод только раздела CPU.

-T, --tasks-only

Не выводить смены состояния процессоров.

-p, --process

Выбор процессов для вывода (по имени или PID)

-f, --force

Выполнять без вопросов.

--symfs=<directory>

Поиск файлов с символами относительно указанного каталога.

-n, --proc-num

Выводить информацию по меньшей мере для указанного числа задач.

-t, --topology

Сортировать CPU в соответствии с топологией.

--highlight=<duration_nsecs|task_name>

Выделять цветом задачи, которые работали дольше указанного времени, или задачи с указанным именем. Если указано число, оно считается продолжительность в наносекундах, нечисловые строки считаются именами задач.

--io-skip-eagain

Не показывать события EAGAIN IO.

--io-min-time=<nsecs>

Выводить мелкие события, как будто они имели минимальную длительность. Это полезно для мелких и коротких операций ввода-вывода. Можно задать время в миллисекундах (суффикс ms) или микросекундах (us). По умолчанию используется 1000000 нсек (1 мсек).

--io-merge-dist=<nsecs>

Объединять события, разделённые интервалом merge-dist нсек. Это снижает число фигур на рисунке SVG и делает его более читаемым. Можно задать время в миллисекундах (суффикс ms) или микросекундах (us). По умолчанию используется 1000 нсек (1 мксек).

Опции режима record

-P, --power-only

Записывать только связанные с питанием события.

-T, --tasks-only

Записывать только связанные с задачами события.

-I, --io-only

Записывать только связанные с вводом-выводом события.

[1]Scalable Vector Graphics - масштабируемая векторная графика.

-g, --callchain
Записывать граф вызовов.

Примеры

```
$ perf timechart record git pull

          [ perf record: Woken up 13 times to write data ]
          [ perf record: Captured and wrote 4.253 MB perf.data (~185801 samples) ]

$ perf timechart

          Written 10.2 seconds of trace to output.svg.
```

Запись временного графика для системы в целом

```
          $ perf timechart record
```

затем генерация timechart с выделением задач gcc

```
          $ perf timechart --highlight gcc
```

Запись событий ввода-вывода в масштабе системы

```
          $ perf timechart record -I
```

затем генерация timechart

```
          $ perf timechart
```

mem

Выводит профиль доступа к памяти. Команда perf mem record выполняет указанную за ней команду и собирает данные в файл perf.data. Могут применяться опции perf record. Команда perf mem report выводит результат профилирования, вызывая perf report с нужным набором опций для вывода профиля доступа к памяти. По умолчанию делаются выборки load и store. Опция -t позволяет выбрать один из вариантов.

В системах Intel выводится задержка использования, а не чистая задержка load или store. Задержка использования включает задержки в очередях, а не только задержки в подсистеме памяти.

Синтаксис

```
perf mem [<options>] (record [<command>] | report)
```

Опции

<command>...
Любая команда, понятная командному процессору.
-i, --input=<file>
Имя входного файла.
-f, --force
Не проверять владение.
-t, --type=<type>
Задаёт тип операции в памяти - load или store (по умолчанию оба - load,store).
-D, --dump-raw-samples
Дамп необработанных выборок в формате, пригодном для анализа, по одной выборке в строке.
-x, --field-separator=<separator>
Задаёт разделитель полей для вывода необработанного дампа (опция -D). По умолчанию пробел.
-C, --cpu=<cpu>
Сообщать только о выборках для указанных CPU. Процессоры можно указать списком через запятые (0,1) или диапазоном (0-2) без пробелов. По умолчанию учитываются все CPU.
-U, --hide-unresolved
Выводить только записи, преобразованные в символы.
-p, --phys-data
Физические адреса выборки при записи/отчёте.

Опции записи

-e, --event <event>
Селектор событий. Для просмотра списка доступных событий служит команда perf mem record -e list.
-K, --all-kernel
Настроить все используемые события на работу в пространстве ядра.
-U, --all-user
Настроить все используемые события на работу в пользовательском пространстве.
-v, --verbose
Подробный вывод (ошибки открытия счётчиков и т. п.).
--ldlat <n>
Указать желаемую загрузку для событий load. (только x86).

Кроме того можно использовать все опции perf report, а для записи - опции perf record.

c2c

Анализатор данных C2C[1] и HITM, позволяющий отслеживать «соперничество» при кэшировании.

В системах x86 инструмент работает на основе задержек загрузки и событий хранения, предоставляемых процессорами Intel. В системах PowerPC используются выборки случайных инструкций с использованием порога. Эти события обеспечивают адрес доступа к памяти, тип доступа (load и store), задержку (в машинных тактах) доступа для загрузки (load).

[1]Cache To Cache - из кэша в кэш.

Инструмент c2c позволяет записывать эти данные и выводить отчёт о деталях доступа для строк кэширования с максимальным числом столкновения (наибольшим числом обращений HITM). Дополнительную информацию о работе с инструментами c2c можно найти по ссылке https://joemario.github.io/blog/2016/09/01/c2c-blog/.

Синтаксис

```
perf c2c record [<options>] <command>
perf c2c record [<options>] - [<record command options>] <command>
perf c2c report [<options>]
```

Опции record

-e, --event=
Выбирает событие PMU. Список доступных событий выводится по команде perf mem record -e list.
-v, --verbose
Подробный вывод (ошибки открытия счётчиков и т. п.).
-l, --ldlat
Измерение задержки load (только x86).
-k, --all-kernel
Настройка всех используемых событий на работу в пространстве ядра.
-u, --all-user
Настройка всех используемых событий на работу в пользовательском пространстве.

Опции report

-k, --vmlinux=<file>
Путь к файлу vmlinux.
-v, --verbose
Подробный вывод (ошибки открытия счётчиков и т. п.).
-i, --input
Задаёт входной файл для обработки.
-N, --node-info
Выводить в отчёте. дополнительную информацию об узде (см. Информация об узле).
-c, --coalesce
Задаёт сортировку полей для одного вывода cacheline. Доступны поля tid, pid, iaddr, dso (см. Объединение).
-g, --call-graph
Задаёт параметры цепочек вызовов (см. perf report).
--stdio
Задаёт вывод в stdio (см. ниже).
--stats
Выводить только таблицы статистики и форсировать режим stdio.
--full-symbols
Выводить символы в полном размере.
--no-source
Не выводить столбец Source:Line.
--show-all
Показывать все собранные строки HITM независимо от порога 0,0005 %.
-f, --force
Не проверять принадлежность.
-d, --display
Переключиться на тип HITM (rmt, lcl) для вывода и сортировки.

Запись C2C

Команда perf c2c record устанавливает опции, связанные с анализом HITM cacheline и вызывает команду perf record.

По умолчанию применяются опции, показанные ниже.

```
-W,-d,--phys-data,--sample-cpu
```

Если опция -e не задаёт иного, в x86 отслеживаются события

```
cpu/mem-loads,ldlat=30/P
cpu/mem-stores/P
```

а в PowerPC

```
cpu/mem-loads/
cpu/mem-stores/
```

Пользователь может задать любые опции perf record после маркера --, например,

```
$ perf c2c record -- -g -a
```

Опции записи описаны выше.

Отчёт C2C

Команда perf c2c report выводит результаты анализа и может работать в двух режимах - stdio и TUI (по умолчанию). Сначала все данные сортируются по адресам cacheline, затем сохраняются детали доступа для каждой строки кэша и далее строки кэша сортируются с учётом пользовательских настроек.

В общем случае вывод содержит два базовых представления - список самых «дорогих» cacheline и детали смещения для каждой строки кэша.

В первом представлении для каждой cacheline в обоих режимах выводятся перечисленные ниже данные.

Index
Индекс (отсчёт с 0) cacheline.

Cacheline
Адрес cacheline (шестнадцатеричный).
Total records
Суммарное число обращений ко всем cacheline.
Rmt/Lcl Hitm
Доля данной строки в общем числе удалённых/локальных обращений HITM (в процентах)
LLC Load Hitm - Total, Lcl, Rmt
Общее, локальное и удалённое число load HITM
Store Reference - Total, L1Hit, L1Miss
Total - все обращения для записи (store);
L1Hit - записи, затрагивающие L1;
L1Miss - записи, не затрагивающие L1.
Load Dram
Число локальных и удалённых обращений к DRAM.
LLC Ld Miss
Общее число обращений с отсутствующим LLC[1].
Total Loads
Суммарное число обращений load.
Core Load Hit - FB, L1, L2
Счётчик обращений load в FB[2] кэша L1 и L2.
LLC Load Hit - Llc, Rmt
Счётчик обращений LLC и Remote.
Во втором представлении выводятся перечисленные ниже данные.

HITM - Rmt, Lcl
Доля (в %) обращений Remote/Local HITM для данного смещение в cacheline.
Store Refs - L1 Hit, L1 Miss
Доля (в %) обращений с наличием и отсутствием записи в кэше L1 для данного смещение в cacheline.
Data address - Offset
Адрес смещения.
Pid
Идентификатор процесса, отвечающего за обращение.
Tid
Идентификатор потока, отвечающего за обращение.
Code address
Адрес кода, ответственного за обращение.
cycles - rmt hitm, lcl hitm, load
Сумма машинных тактов для обращений Remote/Local HITM и базовых load.
cpu cnt
Число процессоров, участвовавших в доступе.
Symbol
Символ кода, связанный со значением Code address.
Shared Object
Имя общего объекта, связанного со значением Code address.
Source:Line
Информация источника, относящаяся к значению Code address.
Node
Узлы, участвующие в обращении (см. Информация об узле).

Информация об узле

Поле Node указывает узлы, к которым был доступ по данному смещению cacheline. Вывод представляется в 3 варианта - идентификаторы узлов через запятые, идентификаторы узлов со статистикой для каждого в формате Node{cpus %hitms %stores}, идентификаторы узлов со список затронутых CPU в формате Node{cpu list}. Пользователь может выбрать вариант с помощью опции -N или клавиши n в интерактивном режиме TUI.

Объединение

Пользователь может задать способ сортировки смещения для cacheline. Ниже перечислены поля, доступные для управления выходными полями смещения caheline:

tid	объединение по TID;
pid	объединение по PID;
iaddr	объединение по адресам кода с выводом полей Code address, Code symbol, Shared Object, Source line;
dso	объединение по общим объектам.

По умолчанию объединение происходит по значениям поля pid,iaddr.

Вывод STDIO

В режиме stdio результаты отображаются на стандартном устройстве вывода и включают перечисленные таблицы.

Trace Event Information
Общая статистика обращений к памяти.
Global Shared Cache Line Event Information
Общая статистика разделяемых cacheline.

[1]Last level Cache - кэш последнего уровня.
[2]Fill Buffer - заполнение буфера.

Shared Data Cache Line Table
Список наиболее «дорогих» cacheline.
Shared Cache Line Distribution Pareto
Список всех смещений доступа для каждой cacheline.

Вывод TUI
В режиме TUI обеспечивается интерактивный интерфейс для работы со списком cacheline и вывода деталей смещения. Справку о работе с интерфейсом можно получить, нажав клавишу ?.

kmem
Инструмент для трассировки и измерения производительности операций с памятью в ядре. Работает в 2 вариантах.

perf kmem record <command>
Запись событий kmem при выполнении произвольной задачи[1].
perf kmem stat
Статистический отчёт о работе памяти ядра.
Синтаксис

```
perf kmem {record|stat} [<options>]
```

Опции
-i <file>, --input=<file>
Задаёт входной файл (по умолчанию perf.data, если stdin не является fifo).
-f, --force
Не проверять принадлежность.
-v, --verbose
Подробный вывод (адреса символов и т. п.).
--caller
Вывод статистики по callsite.
--alloc
Вывод статистики по назначениям (allocation).
-s <key[,key2...]>, --sort=<key[,key2...]>
Сортировка вывода (по умолчанию frag,hit,bytes для slab и bytes, hit для страницы). Возможна сортировка по ключам ptr, callsite, bytes, hit, pingpong, frag для slab и page, callsite, bytes, hit, order, migtype, gfp для страницы. Перед этой опцией следует указывать одну из опций выбора режима, т. е. --slab, --page, --alloc и/или --caller.
-l <num>, --line=<num>
Выводить только num строк.
--raw-ip
Выводить raw ip вместо символов.
--slab
Анализ событий распределителя SLAB.
--page
Анализ событий распределителя страниц.
--live
Показывать страницу статистики «вживую». Выводится общая статистика принятая по умолчанию, но с показом распределенных в текущий момент страниц. Требуется установка опции --page.
--time=<start>,<stop>
Анализ выборок из временного окна <start>,<stop>. Время указывается в формате секунды.микросекунды. Если начало не указано (,x.y), анализ выполняется от начала файла, при опущенном времени завершения (x.y,) анализ выполняется до конца файла.

lock
Анализ блокировок в системе.

perf lock record <command>
Записывает события блокировок от начала до завершение <command>, создавая файл perf.data с записью событий блокировки.
perf lock report
Статистический отчёт.
perf lock script
Вывод необработанных событий блокировки.
perf lock info
Вывод метаданных (потоки или адреса экземпляров блокировки).
Синтаксис

```
perf lock {record|report|script|info}
```

Опции
-i, --input=<file>
Имя входного файла (по умолчанию perf.data, если stdin не является fifo).
-v, --verbose
Подробный вывод (адреса символов и т. п.).
-D, --dump-raw-trace
Дамп необработанной трассировки в формате ASCII.
-f, --force
Выполнять без лишних вопросов.

[1]В этом режиме perf не может корректно обрабатывать опции строки <command>, воспринимая их как свои опции. Использование двойных и одинарных кавычек не решает проблему.

Опции записи

-k, --key=<value>
Ключ сортировки - acquired (используется по умолчанию), contended, avg_wait, wait_total, wait_max, wait_min.

Опции отчёта

-t, --threads
Дамп списка протоколов в perf.data.
-m, --map
Дамп отображения экземпляров блокировки (address:name table)

sched

Инструмент для трассировки и измерения свойств (задержки) планировщиков. Команда поддерживает несколько режимов, описанных ниже.

perf sched record <command>
Запись событий планирования для произвольной задачи.
perf sched latency
Отчёт о задержках планирования по задачам и другие свойства планирования при выполнении задания.
perf sched script
Просмотр подробной трассировки задачи, которая была записана (в настоящее время псевдоним perf script).
perf sched replay
Имитация задачи, записанной с помощью perf sched record, путём запуска макетных потоков, имитирующих задачу, на основе записанной трассировки. Эти потоки могут воспроизводить временные характеристики задачи (время работы CPU, картина сна) при её записи и могут многократно повторяться для измерения производительности.
perf sched map
Вывод текстового представления переключения контекста задачи, записанного с помощью команды perf sched record. Выводятся столбцы для отдельных CPU и двухсимвольные сокращения для работающих на CPU задач. Символ * указывает CPU, на котором произошло событие переключения контекста, а точка - бездействующий CPU.
perf sched timehist
Анализ событий планирования.

Синтаксис

```
perf sched {record|latency|map|replay|script|timehist}
```

Опции

-i, --input=<file>
Имя входного файла (по умолчанию perf.data, если stdin не является fifo).
-v, --verbose
Подробный вывод (адреса символов и т. п.).
-D, --dump-raw-trace=
Вывод подробного дампа данных планирования.
-f, --force
Выполнять без лишних вопросов.

Опции perf sched map

--compact
Показывать только активные CPU. Помогает при визуализации в системах с большим числом ядер.
--cpus
Показывать записи действий для указанных CPU.
--color-cpus
Отметить цветом указанные процессоры.
--color-pids
Отметить цветом указанные pid.

Опции perf sched timehist

-k, --vmlinux=<file>
Путь к файлу vmlinux.
--kallsyms=<file>
Путь к файлу kallsyms.
-g, --call-graph
Выводить цепочки вызовов при их наличии (принято по умолчанию).
--max-stack
Максимальное число функций, показываемых в обратной трассировке (по умолчанию 5)
-p=, --pid=
Показывать только события для указанных через запятые идентификаторов процессов.
-t=, --tid=
Показывать только события для указанных через запятые идентификаторов потоков.
-s, --summary
Показывать только сводку событий планирования по потокам с выводом минимального, максимального и среднего времени работы (в секундах) и связанного stddev.
-S, --with-summary
Показывать все события планирования и сводку по потокам с выводом минимального, максимального и среднего времени работы (в секундах) и связанного stddev.
--symfs=<directory>
Поиск файлов с символами относительно указанного каталога.
-V, --cpu-visual
Выводить визуальные подсказки для переключения планирования между CPU - i указывает бездействие (idle), s - события планировщика.

-w, --wakeups
Показывать события пробуждения.

-M, --migrations
Показывать события переноса (migration).

-n, --next
Показывать следующую задачу.

-I, --idle-hist
Показывать только события, связанные с бездействием.

--time
Анализировать только выборки в интервале <start>,<stop>, указанном в формате секунды.микросекунды. Если начало не указано (,x.y) данные анализируются от начала файла. Если не указно время завершения (x.y,), данные анализируются до конца файла.

--state
Показывать состояние задачи при переключении.

Пример применения

```
perf sched record -- sleep 1
perf sched timehist
```

По умолчанию выводятся отдельные события планирования, включая время ожидания (wait time - время между событием sched-out и следующим событием sched-in для задачи), задержку планирования задачи (sch delay - время между пробуждением и началом работы) и время работы задачи (run time).

```
                 time    cpu  task name             wait time  sch delay   run time
                                [tid/pid]               (msec)     (msec)     (msec)
--------------- ------  --------------------  ---------  ---------  ---------
  79371.874569 [0011]  gcc[31949]                  0.014      0.000      1.148
  79371.874591 [0010]  gcc[31951]                  0.000      0.000      0.024
  79371.874603 [0010]  migration/10[59]            3.350      0.004      0.011
  79371.874604 [0011]  <idle>                      1.148      0.000      0.035
  79371.874723 [0005]  <idle>                      0.016      0.000      1.383
  79371.874746 [0005]  gcc[31949]                  0.153      0.078      0.022
...
```

Время указано в формате миллисекунды.микросекунды.

kvm

Инструмент для трассировки и измерения производительности гостевых ОС KVM. Имеется несколько вариантов использования perf kvm, описанных ниже.

perf kvm [options] top <command>
Для генерации и вывода профиля производительности гостевой ОС в реальном масштабе времени при выполнении произвольной задачи.

perf kvm record <command>
Для записи профиля производительности при выполнении произвольной задачи с сохранением данных в файле perf.data. По умолчанию для поведения perf kvm используется опция --guest, поэтому если на входе нет ни --host, ни --guest, файл данных perf будет иметь имя perf.data.guest. Если задана опция --host, файл получит имя perf.data.kvm. Если нужно записать данные в файл perf.data.host, следует указать --host --no-guest. Это поведение конкретно описано ниже.

без указания -> perf.data.guest
--host -> perf.data.kvm
--guest -> perf.data.guest
--host --guest -> perf.data.kvm
--host —no-guest -> perf.data.host

perf kvm report
Для вывода информации профиля производительности, записанного по команде perf kvm record.

perf kvm diff
Для вывода различий производительности между двумя файлами perf.data, полученными с помощью perf record.

perf kvm buildid-list
Для вывода идентификаторов сборки из файла perf.data, которые могут применяться другими инструментами для извлечения пакетов с соответствующими таблицами символов, используемых командой perf report. Поскольку идентификаторы сборки считываются из /sys/kernel/notes в ОС, для получения списка идентификаторов сборки гостевой ОС следует убедиться, что файл perf.data был получен по команде perf kvm record с опцией --guestmount.

perf kvm stat <command>
Для запуска команды и сбора счётчиков статистики.
В частности, perf kvm stat record/report выполняет анализ статистики событий KVM. В настоящее время поддерживаются события vmexit, mmio (только x86) и ioport (только x86). Команда perf kvm stat record <command> записывает события KVM и события между началом и завершением команды <command>.
Эта команда создаёт файл с результатами трассировки событий KVM.

perf kvm stat report
записывает данные статистики, включая время обработки событий, выборки и т. д.

perf kvm stat live
Выводит статистические данные «вживую» (подобно record + report, но с обновлением данных в процессе работы).

Синтаксис

```
perf kvm [--host] [--guest] [--guestmount=<path>
             [--guestkallsyms=<path> --guestmodules=<path> | --guestvmlinux=<path>]]
             {top|record|report|diff|buildid-list} [<options>]
      perf kvm [--host] [--guest] [--guestkallsyms=<path> --guestmodules=<path>
             | --guestvmlinux=<path>] {top|record|report|diff|buildid-list|stat} [<options>]
      'perf kvm stat [record|report|live] [<options>]
```

Опции

-i, --input=<path>
Имя входного файла.

-o, --output=<path>
Имя выходного файла.

--host
Профиль производительности на стороне хоста.

--guest
Профиль производительности на стороне гостевой системы.

--guestmount=<path>
Каталог монтирования корня файловой системы гостевой ОС. Пользователи монтируют корень гостевой файловой системы по пути <path>, указывая метод доступа к файловой системе (обычно sshfs). Рассмотрим для примера две гостевых ОС, одна из которых имеет pid 8888, другая - 9999.

```
#mkdir /guestmount; cd/guestmount
#sshfs -o allow_other,direct_io -p 5551 localhost:/ 8888/
#sshfs -o allow_other,direct_io -p 5552 localhost:/ 9999/
#perf kvm --host --guest --guestmount=~/guestmount top
```

--guestkallsyms=<path>
Копия файла гостевой ОС /proc/kallsyms. Команда perf kvm читает его для получения символов гостевого ядра. Пользователи копируют это из гостевой ОС.

--guestmodules=<path>
Копия файла гостевой ОС /proc/modules. Команда perf kvm читает его для получения информации модулей ядра. Пользователи копируют это из гостевой ОС.

--guestvmlinux=<path>
Файл vmlinux ядра гостевой ОС.

-v, --verbose
Подробный вывод (ошибки открытия счётчиков и т .п.).

Опции статистического отчёта

--vcpu=<value>
Анализ событий на заданных виртуальных процессорах (по умолчанию все vcpu).

--event=<value>
Событие для анализа - vmexit, mmio (только x86), ioport (только x86). По умолчанию vmexit.

-k, --key=<value>
Ключ сортировки - sample (используется по умолчанию, по номеру выборки), time (по среднему времени).

-p, --pid=
Анализ событий только для указанных через запятые идентификаторов процессов.

Опции статистики “вживую”

-d, --display
Интервал обновления в секундах.

-m, --mmap-pages=
Число страниц данных mmap (должно быть степенью 2) или размер с суффиксом B/K/M/G (округляется до ближайшей степени 2).

-a, --all-cpus
Сбор данных от всех CPU.

-p, --pid=
Анализ событий лишь для указанных через запятые идентификаторов процессов.

--vcpu=<value>
Анализ событий на указанном виртуальном процессоре (по умолчанию учитываются все vcpu).

--event=<value>
Событие для анализа - vmexit, mmio (только x86), ioport (только x86). По умолчанию vmexit.

-k, --key=<value>
Ключ сортировки - sample (принят по умолчанию, сортировка по числу выборок), time (по среднему времени).

--duration=<value>
Показывать события, отличные от HLT (только x86) или Wait (только s390) с продолжительностью больше указанного значения.

--proc-map-timeout
Обработка имеющихся потоков /proc/XXX/mmap может занимать продолжительное время. Эта опция позволяет задать тайм-аут для такой обработки (по умолчанию 500 мсек).

diff

Команда обеспечивает сравнение двух и более файлов perf.data и выводит различия между результатами, собранными с помощью команд perf record.

Если файлы не заданы, сравниваются perf.data.old и perf.data.

Дифференциальный профиль выводится только для событий, присутствующих в обоих файлах perf.data.

Если параметры сортировки не заданы, вывод сортируется по значениям dso и symbol. Поскольку файлы perf.data могут происходить от разных двоичных файлов, адреса символов могут различаться. Поэтому perf diff сравнивает имена.

Синтаксис

```
perf diff [baseline file] [data file1] [[data file2] ... ]
```

Опции

-D, --dump-raw-trace
Дамп необработанной (raw) трассировки в формате ASCII.

--kallsyms=<file>
Путь к файлу kallsyms.

-m, --modules
Загружать символы модуля (работает только с опцией -k и «живым» ядром).

-d, --dsos=
Учитываются символы лишь указанных DSO. Можно использовать файл CSV, заданный параметром file://filename. Эта опция будет влиять на значения процентов в столбцах Baseline/Delta (см. --percentage).

-C, --comms=
Учитываются лишь символы в указанных столбцах. Можно использовать файл CSV, заданный параметром file://filename. Эта опция будет влиять на значения процентов в столбцах Baseline/Delta (см. --percentage).

-S, --symbols=
Учитываются лишь указанные параметром символы. Можно использовать файл CSV, заданный параметром file://filename. Эта опция будет влиять на значения процентов в столбцах Baseline/Delta (см. --percentage).

-s, --sort=
Сортировать по ключам - pid, comm, dso, symbol, cpu, parent, srcline (см. одноимённую опцию команды perf report).

-t, --field-separator=
Задаёт символ-разделитель вместо заполнения пробелами с заменой этого символа в именах (и других полях) символом точки, которая не может служить разделителем.

-v, --verbose
Подробный вывод (например, необработанные счётчики в дополнение к разнице).

-q, --quiet
Не выводить сообщений (отменяет -v).

-f, --force
Не проверять принадлежность.

--symfs=<directory>
искать файлы с символами относительно указанного каталога.

-b, --baseline-only
Показывать только элементы, соответствующие базовой линии.

-c, --compute
Выбор метода расчёта различий - delta (приращение), ratio (отношение), wdiff, delta-abs (абсолютное приращение, принято по умолчанию). Принятое по умолчанию поведение можно изменить в конфигурационной переменной diff.compute. Методы сравнения описаны ниже.

-p, --period
Показывать значения периода для обеих сравниваемых записей.

-F, --formula
Показывать формулу расчёта.

-o, --order
Задаёт номер столбца для сортировки расчёта. Значение 0 задаёт сортировку по базовой линии издержек, а 1 (принято по умолчанию) по рассчитанным значениям столбца 1 (данные из первого файла, отличные от базовых). Значения больше 1 могут использоваться только при достаточном числе файлов. Принятое по умолчанию значение можно изменить в переменной diff.order.

--percentage
Задаёт способ отображения процента издержек для отфильтрованных записей. Фильтры можно задать опциями --comms, --dsos, --symbols.
Значение relative задаёт вывод относительно отфильтрованных записей и сумма всегда будет составлять 100%. В случае absolute расчёт выполняется относительно исходных значений до и после фильтрации.

--time
Задаёт анализ выборок лишь из временного окна, заданного в процентах - a%/n,b%/m,... или a%-b%,c%-%d,...
Например, для выбора вторых 10% интервала записи можно использовать команду

```
perf diff --time 10%/2
```

Для выбора интервала от 0% до 10% служит команда

```
perf diff --time 0%-10%
```

Например, для выбора первых и вторых 10% интервала записи можно использовать команду

```
perf diff --time 10%/1,10%/2
```

Для выбора интервалов от 0% до 10% и от 30% до 40% служит команда

```
perf diff --time 0%-10%,30%-40%
```

Поддерживается также анализ выборок из временного окна <start>,<stop>. Время указывается в формате секунды.микросекунды. Если время начала не указано (т. е. ,x.y), анализ начинается с начала файла. Если время окончания не указано (т. е. x.y,), анализ выполняется до конца файла. Строка указания времени имеет форму 'a1.b1,c1.d1:a2.b2,c2.d2'. Для разделения временных меток разных файлов perf.data используется двоеточие (:).
Например, для получения временных меток из perf script можно использовать команды

```
perf script -i perf.data.old
  mgen 13940 [000]  3946.361400: ...

perf script -i perf.data
  mgen 13940 [000]  3971.150589 ...

perf diff --time 3946.361400,:3971.150589,
```

Анализируется файл perf.data.old от метки 3946.361400 до конца файла и файл perf.data от метки 3971.150589 до конца файла.

--cpu
Сравнивать только выборки для указанных CPU. Процессоры можно указать списком через запятые (0,1) или диапазоном (0-2) без пробелов. По умолчанию учитываются все CPU.

--pid=
Сравнивать только выборки для указанных через запятые идентификаторов процессов.

--tid=
Сравнивать только выборки для указанных через запятые идентификаторов потоков.

Сравнение

Сравнение выполняется с базовым файлом, из которого последовательно считываются выборки. Для этих выборок выполняется поиск соответствия во всех остальных файлах perf.data, указанных в команде. Если парная запись найдена, выполняются указанные расчёты и выводится результат.

Все выборки в файлах perf.data, не соответствующие выборкам в базовом файле, отображаются с незаполненным столбцом baseline и могут включать результаты расчёта (приращение) в своём столбце.

Рассмотрим пример, где файл A включает выборки f1, f2, f3, f4, f6, файл B - выборки f2, f4, f5, а файл C - f1, f2, f5. Ниже приведены возможные варианты вывода для разных команд.

```
-   perf diff A B C
          baseline/A compute/B compute/C  samples
          ----------------------------------------
          b                    x          f1
          b          x         x          f2
          b                               f3
          b          x                    f4
          b                               f6
                     x         x          f5
-   perf diff B A C
          baseline/B compute/A compute/C  samples
          ----------------------------------------
          b          x         x          f2
          b          x                    f4
          b                    x          f5
                     x         x          f1
                     x                    f3
                     x                    f6
-   perf diff C B A
          baseline/C compute/B compute/A  samples
          ----------------------------------------
          b                    x          f1
          b          x         x          f2
          b          x                    f5
                               x          f3
                     x         x          f4
                               x          f6
```

Методы сравнения

delta

В этом случае выводится столбец Delta, значения которого вычисляются как

```
d = A->period_percent - B->period_percent
```

где A и B - записи из сравниваемого и базового файла, period_percent указывает процентную долю записи в одном файле данных. При фильтрации с использованием -C, -d и/или -S, значение period_percent может меняться в результате фильтрации. Предотвратить отклонения можно с помощью опции --percentage=absolute.

delta-abs

Работает как режим delta, но результаты сортируются по абсолютным значениям.

ratio

В этом режиме выводится столбец Ratio, значения которого вычисляются как

```
r = A->period / B->period
```

где A и B - записи из сравниваемого и базового файла, period - значение периода в записи.

wdiff:WEIGHT-B,WEIGHT-A

В этом режиме выводится столбец Weighted diff, значения которого рассчитываются как

```
d = B->period * WEIGHT-A - A->period * WEIGHT-B
```

где A и B - записи из сравниваемого и базового файла, period - значение периода в записи, WEIGHT-A и WEIGHT-B - представленные пользователем веса в опции -c после разделителя вида -c wdiff:1,2.

cycles

В этом режиме выводится столбец [Program Block Range] Cycles Diff с разницей числа машинных тактов для одинаковых программных блоков в файлах. Базовым блоком программы считается код между двумя ветвлениями. [Program Block Range] указывает диапазон базового программного блока. При доступности номеров строк выводятся эти номера, в остальных случаях - symbol+offset.

probe

Определяет новые динамические точки трассировки по символам и регистрам (без отладочной информации) или по выражениям C (номера строк, функции и локальные переменные C) при наличии отладочной информации.

Синтаксис

```
perf probe [options] --add=PROBE […]
perf probe [options] PROBE
perf probe [options] --del=[GROUP:]EVENT [...]
perf probe --list[=[GROUP:]EVENT]
perf probe [options] --line=LINE
perf probe [options] --vars=PROBEPOINT
perf probe [options] --funcs
perf probe [options] --definition=PROBE [...]
```

Опции

-k, --vmlinux=PATH

Задаёт путь к файлу vmlinux с отладочной информацией (dwarf). При использовании вместе с опцией --definition можно указать неактивный (offline) файл vmlinux.

-m, --module=MODNAME|PATH

Задаёт имя модуля, где perf probe будет искать точки или линии для зондов. Если путь пропущен, perf probe считает модуль неактивным (offline), это позволяет добавлять зонды в модуль, который ещё не загружен.

-s, --source=PATH
Задаёт путь к исходным кодам ядра.
-v, --verbose
Подробный вывод (показ анализируемых аргументов и т. п.). Не может применяться вместе с опцией -q.
-q, --quiet
Сокращённый вывод (без показа каких-либо сообщений, включая ошибки). Не совместима с опцией -v.
-a, --add=
Определяет событие зонда (см. Синтаксис зондов).
-d, --del=
Удаляет события зондов. Могут применяться шаблоны glob (*, ?) и классы символов (например, [a-z], [!A-Z]).
-l, --list[=[GROUP:]EVENT]
Выводит список текущих событий зондов. Здесь можно указывать шаблоны фильтрации имён. При использовании с опцией --cache будут выводиться кэшированные зонды вместо активных.
-L, --line=
Показывает строки исходного кода, которые будут проверяться. Для этой опции нужен аргумент, указывающий диапазон строк исходного кода (см. Синтаксис строк).
-V, --vars=
Показывает доступные локальные переменные точки зондирования. Синтаксис аргументом совпадает с описанным ниже, но без ARG.
--externs
Вывод определённых вовне переменных в дополнение к локальным (только для --vars).
--no-inlines
(только для --add) Поиск только функций, не являющихся встроенными (inline). Функции, не имеющие экземпляров, игнорируются.
-F, --funcs[=FILTER]
Показывать доступные функции данного модуля или ядра. При наличии опции -x или --exec могут также перечисляться функции в пользовательском пространстве исполняемого файла или общей библиотеки. Могут также применяться правила фильтрации (FILTER).
-D, --definition=
Показывать определение события трассировки, полученное из данного события зондирования, вместо записи в tracing/[k,u]probe_events.
--filter=FILTER
(только для --vars и --funcs) Задаёт фильтр (FILTER) из шаблонов glob (см. Шаблоны фильтров). По умолчанию применяется фильтр "!k???tab_* & !crc_*" для --vars и "!_*" для --funcs. При указании нескольких фильтров применяется последний.
-f, --force
Принудительно добавлять события с имеющимся именем.
-n, --dry-run
Холостой прогон. С этой опцией операции --add и --del не ведут к реальному добавлению или удалению.
--cache
Кэшировать зонды (с --add) - все добавленные события будут сохраняться в кэш-файле.
Показывать кэшированные зонды (с --list). Удалить кэшированные зонды (с --del).
--max-probes=NUM
Задаёт максимальное число точек отслеживания для события (по умолчанию 128).
--target-ns=PID
Получить информацию о пространстве имён. монтирования от целевого pid. Это применяется при создании зонда для процесса размещённого в другом пространстве имён. монтирования, нежели утилита perf.
-x, --exec=PATH
Задаёт путь к исполняемому файлу или общей библиотеке для трассировки в пользовательском пространстве. Может использоваться с опцией --funcs.
--demangle
Восстанавливать (demangle) символы приложения. Для отмены может использоваться опция --no-demangle.
--demangle-kernel
Восстанавливать (demangle) символы ядра. Для отмены может использоваться опция --no-demangle-kernel.

При отсутствии опций -m/-x команда perf probe проверяет первый аргумент после опций. Если это абсолютный путь, perf probe использует его в качестве цели для зонда пользовательского пространства или модуля.

Синтаксис зондов

Точки зондирования определяются с использованием показанного ниже синтаксиса.

1) По имени функции

```
[[GROUP:]EVENT=]FUNC[@SRC][:RLN|+OFFS|%return|;PTN] [ARG ...]
```

2) По файлу и номеру строки исходного кода

```
[[GROUP:]EVENT=]SRC:ALN [ARG ...]
```

3) По файлу исходного кода с шаблоном

```
[[GROUP:]EVENT=]SRC;PTN [ARG ...]
```

4) Предопределённые события SDT[1] или кэшированное событие с именем

```
%[sdt_PROVIDER:]SDTEVENT
```

или

```
sdt_PROVIDER:SDTEVENT
```

Поле EVENT указывает имя нового события и при отсутствии поля будет использовано имя пробной функции, а к возвращаемым пробам автоматически будет добавляться суффикс __return. Можно указать имя группы в поле GROUP, а при его отсутствии будет использоваться установленное имя зонда для kprobe и probe_<bin> - для uprobe.

[1]Statically Defined Tracepoint - заданные статически точки трассировки [4].

Отметим, что при использовании имени существующей группы могут возникать конфликты с другими событиями. В частности, использование имени группы, зарезервированного для модулей ядра, может скрывать встроенные события в модулях. FUNC указывает имя зондируемой функции и может принимать одну из форм: +OFFS указывает смещение от точки входа в функцию в байтах, :RLN указывает номер строки относительно входной строки функции, %return означает зондирование возврата из функции, а ;PTN задаёт шаблон сопоставления (см. Сопоставление с шаблоном). Отметим, что ;PTN должно помещаться в конце определения точки зондирования. Необязательное поле @SRC указывает файл с исходным кодом, где размещена эта функция. Можно также указать точку зондирования номером строки или шаблоном сопоставления, используя синтаксис SRC:ALN или SRC;PTN, где SRC задаёт путь к файлу с исходным кодом, :ALN - номер строки, а ;PTN - шаблон сопоставления. ARG задаёт аргументы точки зондирования (см. Аргументы зонда). SDTEVENT и PROVIDER указывают предопределённое имя события, которое задано пользовательским SDT или кэшированными ранее зондами с именем события. Отметим, что перед использованием события SDT нужно просканировать целевой двоичный файл (где определены события SDT) с помощью perf buildid-cache, чтобы добавить события SDT в кэш.

ESCAPE-символы

С синтаксисе probe символы =, @, +, : и ; имеют специальное значение. Для блокирования специальной трактовки следует использовать перед ними escape-символ \. Это полезно при зондировании символов определённой версии (например, @GLIBC_) или для указания файлов исходного кода с такими символами в имени. Отметим, что символ \ имеет специальное значение в командных процессорах и может потребоваться два таких символа (\\) или использование кавычек 'AAA\@BBB' (см. Примеры).

Аргументы зонда

Каждый аргумент зонда использует приведённый ниже синтаксис

```
[NAME=]LOCALVAR|$retval|%REG|@SYMBOL[:TYPE]
```

NAME задаёт (необязательное) имя аргумента. Можно использовать имя локальной переменной, элемента локальной структуры (например, var→field, var.field2), локальный массив с фиксированным индексом (например, array[1], var→array[0], var→pointer[2]), или формат аргументов трассировщика kprobe (например, $retval, %ax). Отметим, что имя аргумента будет указано в качестве имени последнего элемента, если вы зададите элемент локальной структуры данных (например. field2 для var→field1.field2.) Специальные аргументы $vars и $params также могут применяться для NAME, при этом $vars преобразуется в локальные переменные (включая параметры функции), которые могут быть доступны в данной точке зондирования, $params преобразуется только в параметры функции. TYPE служит для приведения типа аргумента (необязательно) и при отсутствии его perf probe автоматически задаёт тип на основе debuginfo. В настоящее время поддерживаются базовые типы (u8/u16/u32/u64/s8/s16/s32/s64), шестнадцатеричные целые числа (x/x8/x16/x32/x64), приведение к знаку (u/s), string и bitfield (см. Типы). В системах x86 запись %REG всегда означает краткую форму регистра, например, %AX (%RAX или %EAX недействительны).

Типы

Базовые типы (u8/u16/u32/u64/s8/s16/s32/s64) и шестнадцатеричные целые (x8/x16/x32/x64) являются целочисленными типами. Префиксы s и u задают наличие (signed) и отсутствие (unsigned) знака, а x указывает шестнадцатеричный формат. Отслеживаемые аргументы задаются в десятичной (sNN/uNN) или шестнадцатеричной (xNN) форме. Можно использовать префиксы s или u для указания знака, а можно опускать префикс, позволяя perf probe определить его автоматически. Префикс x можно использовать для явного указания шестнадцатеричных чисел (размер определяется автоматически). Тип string служит для завершаемых null-символом строк из пространства ядра. Это означает отказ или сохранение значения NULL, если контейнер строки переходит через границу страницы. Можно указывать тип string лишь для локальной переменной или элемента структуры, который является массивом или указателем на тип char или unsigned char. Другой специальный тип bitfield принимает 3 параметра - размер (bit-width), смещение (bit-offset) и размер контейнера (container-size, обычно 32). Синтаксис имеет вид

```
b<bit-width>@<bit-offset>/<container-size>
```

Синтаксис строк

Диапазон строк описывается приведённым ниже синтаксисом.

```
"FUNC[@SRC][:RLN[+NUM|-RLN2]]|SRC[:ALN[+NUM|-ALN2]]"
```

FUNC задаёт имя функции для показанных строк. RLN указывает номер первой строки от точки входа в функцию, RLN2 - номер последней строки. Как и для синтаксиса проб SRC указывает путь к файлу исходного кода, ALN - номер стартовой строки, ALN2 — номер последней строки в файле. Можно также задать число выводимых строк параметром NUM. Кроме того, комбинация FUNC@SRC удобна для поиска конкретной функции среди нескольких одноимённых. Так "source.c:100-120" показывает строки с 100-й по l20-ю в файле source.c, а "func:10+20" - 20 строк, начиная с 10-й, для функции func.

Сопоставление с шаблоном

«Ленивое» сопоставление строк похоже на сопоставление glob, но игнорирует пробелы в обоих сравниваемых частях. Для сопоставления можно применять символы-шаблоны (*, ?) и классы символов (например, [a-z], [!A-Z]). Например, выражению a=* будут соответствовать a=b, a = b, a == b и т. п..

Это обеспечивает некоторую гибкость и отказоустойчивость определений точек зондирования к незначительному изменению кода. Например, 10-я строка schedule() может быть перемещена при изменении schedule(), но строка, соответствующая rq=cpu_rq*, может сохраниться в функции.

Шаблоны фильтров

Шаблон фильтра - это выражения glob для сопоставления с переменными фильтра. В дополнение можно использовать символ ! Для обращения правила. Можно задать несколько правил, объединённых операциями & (и) и | (или), и «свернуть» эти правила в одно с помощью скобок. Например с --filter "foo* | bar*", команда perf probe -V покажет переменные, начинающиеся с foo или bar. С фильтром --filter "!foo* & *bar" команда perf probe -V покажет переменные, которые начинаются не с foo и заканчиваются на bar, такие как fizzbar. Но foobar будет отфильтрована.

Примеры

Вывод строк в schedule(), которые могут быть прозондированы

```
./perf probe --line schedule
```

Добавление зонда в 12-ю строку функции schedule() function с записью локальной переменной cpu

```
./perf probe schedule:12 cpu
```

или

```
./perf probe --add='schedule:12 cpu'
```

Добавление одного или нескольких зондов, имена которых начинаются с schedule

```
./perf probe schedule*
```

или

```
./perf probe --add='schedule*'
```

Добавление зондов в строки функции schedule(), вызывающие update_rq_clock()

```
./perf probe 'schedule;update_rq_clock*'
```

или

```
./perf probe --add='schedule;update_rq_clock*'
```

Удаление всех зондов из schedule()

```
./perf probe --del='schedule*'
```

Добавление зондов в функцию zfree() командного процессора /bin/zsh

```
./perf probe -x /bin/zsh zfree or ./perf probe /bin/zsh zfree
```

Добавление зондов в функцию malloc() библиотеки libc

```
./perf probe -x /lib/libc.so.6 malloc or ./perf probe /lib/libc.so.6 malloc
```

Добавление uprobe в целевой процесс, работающий в другом пространстве имён. (монтирования)

```
./perf probe --target-ns <target pid> -x /lib64/libc.so.6 malloc
```

Добавление зонда USDT в целевой процесс, работающий в другом пространстве имён. (монтирования)

```
./perf probe --target-ns <target pid> -x
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.el7_3.x86_64/jre/lib/amd64/server/libjvm.so
%sdt_hotspot:thread__sleep__end
```

Добавление зонда в символ конкретной версии с использованием escape

```
./perf probe -x /lib64/libc-2.25.so 'malloc_get_state\@GLIBC_2.2.5'
```

Добавление зонда в файл исходного кода с использованием escape-символа

```
./perf probe -x /opt/test/a.out 'foo\+bar.c:4'
```

inject

Фильтр для добавления информации в поток событий. Команда perf inject читает поток событий perf record и передаёт его в stdout. В любой точке код обработки может вставить в поток другие события - в этом случае считываются идентификаторы сборки (опция -b) и помещаются в поток событий.

Идентификаторы сборки являются лишь первым примером использования perf inject, на деле в поток можно добавить все, что требует обработки в пользовательском пространстве, для дополнения потока событий информацией.

Синтаксис

```
perf inject <options>
```

Опции

-b, --build-ids=
Помещает идентификаторы сборки в поток событий.

-v, --verbose
Подробный вывод.

-i, --input=
Имя входного файла (по умолчанию stdin).

-o, --output=
Имя выходного файла (по умолчанию stdout).

-s, --sched-stat
Объединяет sched_stat и sched_switch для получения событий о точках (цепочка вызовов) и продолжительности сна задачи.

--kallsyms=<file>
путь к файлу kallsyms.

--itrace
Задаёт опции декодирования данных трассировки инструкций, заменяемые синтезированными событиями:

- i синтезировать события инструкций;
- b синтезировать события ветвления;
- c синтезировать события ветвления (только вызовы);
- r синтезировать события ветвления (только возвраты);
- x синтезировать события транзакций;
- w синтезировать события ptwrite;
- p синтезировать события, связанные с питанием;
- e синтезировать события, связанные с ошибками;
- d создать журнал отладки;
- g синтезировать цепочку вызовов (используется с i или x);
- l синтезировать записи последнего ветвления (используется с i или x);
- s пропустить указанное число записей в начале.

По умолчанию учитываются все события (--itrace=ibxwpe), за исключением perf (--itrace=ce)
В дополнение к этому можно задать интервал учёта (по умолчанию 100000, за исключением perf script, где устанавливается 1) в удобных единицах:

i инструкции;
t такты часов;
ms миллисекунды;
us микросекунды;
ns наносекунды (используется по умолчанию).

Можно также указать размер цепочки (по умолчанию 16, максимум 1024) для событий.
Можно указать также число записей последнего ветвления (по умолчанию 64, максимум 1024).
Можно пропустить генерацию событий (инструкции, ветвления, транзакции, ptwrite, питание) в начале. Это удобно для пропуска фазы инициализации. Например, опция

```
--itrace=i0nss1000000
```

обеспечит пропуск первого миллиона инструкций.

--strip
используется с --itrace для удаления несинтезированных событий.

-j, --jit
Обрабатывать файлы jitdump путём вставки записей mmap, соответствующих jit[1]-функциям. Эта опция также генерирует образы ELF для каждой jit-функции в файлах jitdump, собранных во входном файле perf.data. Опцию следует применять при мониторинге среды, использующей JIT-приложения, такие как Java, DART, V8.

-f, --force
Выполнять без лишних вопросов.

archive

Создаёт архив объектных файлов с идентификаторами сборки, найденными в файле perf.data. Эта команда запускает команду perf buildid-list --with-hits и собирает файлы с найденными идентификаторами сборки для обеспечения возможности анализа perf.data на другой машине.

Синтаксис

```
perf archive [file]
```

data

Преобразование формата файла данных.
Синтаксис

```
perf data [<common options>] <command> [<options>]
```

Команды

convert
Преобразует файл perf.data в другой формат (в настоящее время поддерживается лишь формат CTF [1]). Можно установить переменную --debug для получения отладочных сообщений в процессе преобразования (perf --debug data …).

Опции преобразования

--to-ctf
Задаёт преобразование в формат CTF и путь к каталогу данных CTF.

-i
Задаёт путь к входному файлу данных perf.

-f, --force
Форсирует преобразование без лишних вопросов.

-v, --verbose
Задаёт подробный вывод (ошибки открытия счётчиков и т. п.).

--all
Преобразовывать все события, включая не связанные с выборкой (comm, fork, ...). По умолчанию выключено и преобразуются лишь выборки.

list

Выводит список символьных имён. событий, которые могут выбираться в командах perf с опцией -e.

Синтаксис

```
perf list [--no-desc] [--long-desc] [hw|sw|cache|tracepoint|pmu|sdt|metric|metricgroup|event_glob]
```

Опции

-d, --desc
Выводить дополнительные описания событий (принято по умолчанию).

--no-desc
Не выводить описания событий.

-v, --long-desc
Выводить более подробные описания.

--debug
Включить отладочный вывод.

--details
Выводит информацию о преобразовании именованных событий в события perf, а также дополнительные выражения, рассчитываемые perf stat.

[1]Just-in-time - (компиляция) «на лету».

Модификаторы событий

События могут включать модификаторы, добавленные через двоеточие. Модификаторы позволяют ограничить учитываемые события. Доступные модификаторы перечислены ниже.

- u учёт для пользовательского пространства;
- k учёт для ядра;
- h учёт для гипервизора;
- I учёт действующих (не idle) событий;
- G учёт гостевых событий в KVM;
- H учёт событий хоста (не KVM);
- p точный уровень;
- P использовать максимальный обнаруженный точный уровень;
- S считывать значение в выборке (PERF_SAMPLE_READ);
- D прикрепить событие к PMU;
- W группа задана слабо и при отсутствии планирования не используется.

Модификатор p может применяться для указания степени точности адреса инструкции и может указываться несколько раз:

- 0 SAMPLE_IP может иметь произвольный идентификатор skid;
- 1 SAMPLE_IP должен иметь постоянный идентификатор skid;
- 2 для SAMPLE_IP запрошен skid 0;
- 3 SAMPLE_IP должен иметь skid 0 или быть случайным для предотвращения эффектов «затемнения».

Для систем Intel точная выборка событий реализуется с помощью PEBS[1], поддерживающей уровни 0-2, а в особых случаях и 3.

В системах AMD это реализовано с помощью IBS[2] (до уровня 2). Модификатор precise работает с событиями типов 0x76 (cpu-cycles, часы CPU не остановлены) и 0xC1 (микрооперации скрыты). Оба события отображаются на выборку IBS (IBS op) с соответственно установленным битом IBS Op Counter Control (IbsOpCntCtl) [2]. Примеры использования IBS приведены ниже.

```
perf record -a -e cpu-cycles:p ...    # use ibs op counting cycles
perf record -a -e r076:p ...          # same as -e cpu-cycles:p
perf record -a -e r0C1:p ...          # use ibs op counting micro-ops
```

Дескриптор необработанного аппаратного события

Даже если символьная форма события не доступна в данный момент программе perf, она может кодироваться зависимым от процессора способом.

Например, для процессоров x86 NNN представляет кодирование по схеме IA32_PERFEVTSELx MSR (см. Figure 30-1 Layout of IA32_PERFEVTSELx MSRs в [3]) или AMD PerfEvtSeln (см. Figure 13-7 Performance Event-Select Register (PerfEvtSeln) в [2], стр. 344).

Примечание. В регистрах счётчиков x86 можно установить только поля event, umask, edge, inv, cmask. В частности, флаги режимов guest/host и OS/user должны устанавливаться только с помощью модификаторов событий (см. выше).

Например, если в документации Intel для QM720 Core i7 событие описано, как показано ниже.

```
Event  Umask  Event Mask
Num.   Value  Mnemonic    Description                          Comment

A8H      01H  LSD.UOPS    Counts the number of micro-ops       Use cmask=1 and
                          delivered by loop stream detector    invert to count
                                                               cycles
```

Можно использовать необработанное кодирование в форме

```
perf stat -e r1a8 -a sleep 1
perf record -e r1a8 ...
```

Для получения информации следует обращаться к документации используемого процессора.

Произвольные PMU

Программа perf поддерживает расширенный синтаксис для задания необработанных параметров PMU. Для работы с ним обычно требуется разбираться с конкретным событием по документации производителя CPU.

Список доступных PMU и их необработанных параметров можно получить с помощью команды

```
ls /sys/devices/*/format
```

Например, событие LSD.UOPS можно указать в форме

```
perf stat -e cpu/event=0xa8,umask=0x1,name=LSD.UOPS_CYCLES,cmask=0x1/ ...
```

или с использованием расширенного синтаксиса

```
perf stat -e cpu/event=0xa8,umask=0x1,cmask=0x1,name=\'LSD.UOPS_CYCLES:cmask=0x1\'/ ...
```

[1]Precise Event-Based Sampling - точная выборка на основе событий.
[2]Instruction Based Sampling - выборка на основе инструкций.

PMU на уровне сокета

Некоторые PMU связаны не с ядрами, а с сокетом CPU в целом. События в таких PMU в общем случае невозможно выбрать и можно лишь учитывать глобально по команде perf stat -a. Они могут быть связаны с одним логическим CPU, но будут измеряться на всех ядрах одного сокета.

Приведённая ниже команда измеряет пропускную способность памяти каждую секунду на первом контроллере сокета 0 в системе Intel Xeon

```
perf stat -C 0 -a uncore_imc_0/cas_count_read/,uncore_imc_0/cas_count_write/ -I 1000 ...
```

Каждый контроллер памяти имеет свой блок PMU. Измерение полной пропускной способности в системе потребует указания всех imc PMU (см. вывод perf list) и сложения полученных значений. Для упрощения создания множества событий, префиксов и шаблонов имён. PMU подстроки uncore_ игнорируются при сопоставлении. Поэтому приведённую выше команду можно распространить на все контроллеры, используя показанный ниже синтаксис

```
perf stat -C 0 -a imc/cas_count_read/,imc/cas_count_write/ -I 1000 ...
perf stat -C 0 -a *imc*/cas_count_read/,*imc*/cas_count_write/ -I 1000 ...
```

Приведённый ниже пример служит для измерения суммарной потребляемой энергии каждую секунду

```
perf stat -I 1000 -e power/energy-cores/ -a
```

Ограничение доступа

Обычным пользователям (не root) обычно доступны лишь события PMU для переключения контекста. Это обычно лишь предопределённые события в PMU, такие как циклы, инструкции и некоторые программные события.

Другие PMU и глобальные измерения обычно доступны лишь пользователю root. Некоторые классификаторы событий (такие как any) тоже доступны только для root.

Это поведение можно изменить путём установки для переменной sysctl kernel.perf_event_paranoid значения -1, позволяющего обычным пользователям видеть такие события.

Для доступа к событиям точек трассировки программе perf нужно читать каталог /sys/kernel/debug/tracing даже при смягчённой установке perf_event_paranoid.

Аппаратная трассировка

Некоторые PMU контролируют расширенные возможности аппаратной трассировки (такие как Intel PT), что позволяет выполнять трассировку с меньшими издержками. Описание Intel PT приведено в отдельном документе intel-pt.txt.

Параметризованные события

Некоторые события PMU, сообщаемые по команде perf list, будут указаны со знаком вопроса (?). Например,

```
hv_gpci/dtbp_ptitc,phys_processor_idx=?/
```

Это означает, что предоставление этого события должно сопровождаться представлением значения для ?. Например,

```
perf stat -C 0 -e 'hv_gpci/dtbp_ptitc,phys_processor_idx=0x2/' ...
```

Классификаторы событий

Можно также добавлять к событиям классификаторы. Например

```
percore:
```

Для суммирования счётчиков событий во всех аппаратных потоках ядра можно ввести команду

```
perf stat -e cpu/event=0,umask=0x3,percore=1/
```

Группы событий

Perf поддерживает мультиплексирование событий по времени, когда число активных событий превышает число счётчиков производительности оборудования. Мультиплексирование может вызывать ошибки измерения при изменении профиля выполнения рабочей нагрузки.

При расчёте параметров с использованием формул на основе числа событий полезно убедиться в том, что некоторые события всегда измеряются как группа для минимизации ошибок мультиплексирования. События группируются с помощью фигурных скобок { }.

```
perf stat -e '{instructions,cycles}' ...
```

Число доступных счётчиков производительности зависит от CPU. Число событий в группе не может быть больше числа доступных счётчиков. Например, Intel Core CPU обычно включают 4 базовых счётчика на ядро и 3 фиксированных счётчика для инструкций, циклов и опорных циклов (ref-cycle). Для некоторых событий имеются ограничений на планируемые счётчики и они могут не поддерживаться в нескольких экземплярах для группы. При включении в группу слишком большого числа событий некоторые из них не будут учитываться.

Глобально прикреплённые события могут ограничивать число счётчиков, доступных для других групп. В системах x86 по умолчанию NMI watchdog имеет прикрепленный счётчик. Это можно отключить командой

```
echo 0 > /proc/sys/kernel/nmi_watchdog
```

События из разных PMU нельзя смешивать в одну группу, однако для программных событий имеется ряд исключений.

Лидер выборки

Программа perf поддерживает указание лидера групповой выборки с помощью спецификатора :S.

```
perf record -e '{cycles,instructions}:S' ...
perf report --group
```

Обычно отбираются все события группы, но при использовании :S выборка делается только для первого (лидер), а для остальных лишь считываются значения.

Опции

Без указания опций будут выведены все известные события. Для ограничения списка можно применять перечисленные ниже опции.

1. hw или hardware для вывода списка аппаратных событий (например, cache-misse).
2. sw или software для вывода списка программных событий, таких как переключение контекста.
3. cache или hwcache для вывода событий аппаратного кэширования (например, L1-dcache-load).
4. tracepoint для списка всех событий трассировки. Альтернативой служит subsys_glob:event_glob для фильтрации по подсистемам трассировки, таким как sched, block и т. п..
5. pmu для вывода списка предоставляемых ядром событий PMU.
6. sdt для вывода всех событий SDT.
7. metric для списка параметров метрики.
8. metricgroup для списка групп параметров метрики.
9. Если не подходит ничего из перечисленного выше, можно воспользоваться регулярным выражением (glob) для поиска совпадений в списке всех событий.
10. Можно также применить поиск подстроки в именах всех событий.

Можно указать несколько типов сразу (через пробелы) для вывода соответствующих списков.

Для необработанных форматов служит опция --raw-dump:

1. --raw-dump показывает raw-dump для всех событий;
2. --raw-dump [hw|sw|cache|tracepoint|pmu|event_glob] показывает raw-dump заданного типа событий.

evlist

Вывод списка событий из файла perf.data.

Синтаксис

```
perf evlist <options>
```

Опции

-i, --input=
Имя входного файла (по умолчанию perf.data, если stdin не является fifo).
-f, --force
Форсированные выполнение без лишних вопросов.
-F, --freq=
Показать просто частоту выборки, использованную для каждого события.
-v, --verbose=
Показывать все поля.
-g, --group
Показывать информацию о группах событий.
--trace-fields
Показывать имена полей точек трассировки.

buildid-list

Выводит список идентификаторов сборки (build-id), найденных в файле perf.data, которые позволяют использовать другие инструменты для извлечения пакетов с соответствующими таблицами символов при создании отчётов с помощью perf report.

Команда также может служить для вывода идентификаторов сборки работающего ядра или файла ELF (-i/--input).

Синтаксис

```
perf buildid-list <options>
```

Опции

-H, --with-hits
Показывать только затронутые DSO.
-i, --input=
Имя входного файла (по умолчанию perf.data, если stdin не является fifo).
-f, --force
Не проверять владельцев.
-k, --kernel
Показывать идентификатор сборки работающего ядра.
-v, --verbose
Подробный вывод

buildid-cache

Управляет кэшем идентификаторов сборки. Можно добавлять, удалять, обновлять и исключать записи в кэше. В будущем можно будет задать максимальный размер пространства для хранения кэша и т. п.

Команда также сканирует целевой двоичный файл на предмет точек SDT и записывает их вместе с кэшем идентификаторов сборки для последующего использования командой perf probe.

Синтаксис

```
perf buildid-cache <options>
```

Опции

-a, --add=
Добавляет указанный файл в кэш.

-f, --force
Выполнять без дополнительных вопросов.

-k, --kcore
Добавляет в кэш указанный файл kcore. Для текущего хоста это файл /proc/kcore, для чтения которого нужны полномочия root. Запуск buildid-cache от имени root может обновлять кэш идентификаторов сборки root, а не пользователя. Для просмотра точки создания файла служит опция -v. Отметим, что скопированный файл содержит только секции кода, а не весь образ. Файлы kallsyms и modules должны быть в том же каталоге и тоже копироваться. Все 3 файла создаются с правом чтения только для root. Файл kcore не будет добавлен, если kcore уже есть у кэше (с тем же build-id) и имеет те же модули с теми же адресами. Проверить актуальность kcore можно с помощью опции -v.

-r, --remove=
Удаляет из кэша двоичный файл с тем же build-id.

-p, --purge=
Очищает все кэшированные двоичные файлы, включая старые кэши, имеющие указанный путь.

-P, --purge-all
Очищает все кэшированные двоичные файлы, полностью сбрасывая кэш.

-M, --missing=
Выводит список идентификаторов сборки в кэше для указанного файла.

-u, --update=
Обновляет указанный файл в кэше. Старые записи не удаляются, поскольку они ещё могут быть нужны для аннотирования старых (или удалённых) файлов perf.data. Заменяется лишь кэшированный файл с тем же идентификатором сборки. Это может применяться для обновления kallsyms и dso ядра до уровня vmlinux с целью поддержки аннотирования.

-l, --list
Список всех действительных двоичных файлов в кэше.

-v, --verbose
Подробный вывод.

--target-ns=PID
Получить информацию о пространстве имён. монтирования от целевого pid. Это применяется при создании зонда для процесса размещённого в другом пространстве имён. монтирования, нежели утилита perf.

kallsyms

Ищет символы для работающего ядра. Просматривается файл kallsyms и выводится информация о найденных символах, включая DSO, начальный и конечный адрес kallsyms и адреса в таблице символов ELF kallsyms (для символов в модулях).

Синтаксис

```
perf kallsyms [<options>] symbol_name[,symbol_name...]
```

Опции

-v, --verbose=
Расширенный вывод с показом деталей загрузки таблицы символов и т. п.

config

Команда позволяет считывать и устанавливать значения переменных в файле конфигурации.

Синтаксис

```
perf config [<file-option>] [section.name[=value] ...]
perf config [<file-option>] -l | --list
```

Опции

-l, --list
Показывает текущие переменные (имена и значения) из всех разделов конфигурационного файла.

--user
Задаёт имя пользователя для работы с опциями в файле $HOME/.perfconfig.

--system
Служит для работы с системным файлом конфигурации $(sysconfdir)/perfconfig.

Конфигурационный файл

Конфигурационный файл perf содержит множество переменных для управления различными аспектами работы инструментов, включая вывод, использование дисков и т. п. Файлы $HOME/.perfconfig служат для управления конфигурацией отдельных пользователей, а файл $(sysconfdir)/perfconfig служит для задания общих параметров. В современных системах конфигурационных файлов perf может просто не присутствовать, поскольку используемые по умолчанию значения переменных заданы непосредственно в программе. В этом случае вывод команды perf config -l будет пустым.

При считывании или записи переменных используются по умолчанию системный и пользовательский конфигурационные файлы, а опции --system и --user позволяют указать один из этих файлов.

Синтаксис

Конфигурационный файл содержит множество разделов. Каждый раздел начинается именем, указанным в квадратных скобках и продолжается до начала следующего раздела. Каждая переменная должна быть в одном из разделов и задаётся в форме name = value. Например,

```
[section]
        name1 = value1
        name2 = value2
```

Регистр символов в именах разделов учитывается, а имена могут включать любые символы кроме перевода строки. Для символов двойных кавычек и обратной дробной черты применяются escape-последовательности \" и \\. Имя раздела не может включать несколько строк.

Пример

Ниже приведён пример пользовательского файла $HOME/.perfconfig.

```
# # Это пользовательский конфигурационный файл. # и #; указывают комментарии.

[colors]
        # Color variables
        top = red, default
        medium = green, default
        normal = lightgray, default
        selected = white, lightgray
        jump_arrows = blue, default
        addr = magenta, default
        root = white, blue

[tui]
        # Defaults if linked with libslang
        report = on
        annotate = on
        top = on

[buildid]
        # Default, disable using /dev/null
        dir = ~/.debug

[annotate]
        # Defaults
        hide_src_code = false
        use_offset = true
        jump_arrows = true
        show_nr_jumps = false

[help]
        # Format can be man, info, web or html
        format = man
        autocorrect = 0

[ui]
        show-headers = true

[call-graph]
        # fp (framepointer), dwarf
        record-mode = fp
        print-type = graph
        order = caller
        sort-key = function

[report]
        # Defaults
        sort_order = comm,dso,symbol
        percent-limit = 0
        queue-size = 0
        children = true
        group = true

[llvm]
        dump-obj = true
        clang-opt = -g
```

Можно скрыть исходный код при аннотировании с помощью команды

```
% perf config annotate.hide_src_code=true
```

Для добавления или изменения нескольких элементов конфигурации можно воспользоваться командой вида

```
% perf config ui.show-headers=false kmem.default=slab
```

Для изменения порядка сортировки функций отчёта в пользовательском файле (~/.perfconfig) служит команда

```
% perf config --user report sort-order=srcline
```

Для изменения цвета выбранной строки с системном конфигурационной файле ($(sysconf)/perfconfig) служит команда

```
% perf config --system colors.selected=yellow,green
```

Для запроса режима графа вызовов служит команда

```
% perf config call-graph.record-mode
```

Для считывания нескольких пар «ключ-значение» можно использовать команду вида

```
% perf config report.queue-size call-graph.order report.children
```

Для считывания порядка сортировки из пользовательского файла (~/.perfconfig), служит команда

```
% perf config --user call-graph.sort-order
```

Для запроса каталога с идентификаторами сборки в системном файле ($(sysconf)/perfconfig) служит команда

```
% perf config --system buildid.dir
```

Переменные

colors.*

Переменные для управления цветом при выводе команд report, top и annotate в интерфейсе TUI. Следует указывать цвет символов и фона через запятую. Например,

```
medium = green, lightgray
```

Если нужно использовать цвета, настроенные для терминала, просто указывается значение default. Например,

```
medium = default, lightgray
```

Доступны цвета red, yellow, green, cyan, gray, black, blue, white, default, magenta, lightgray.

colors.top

Переменная top задаёт цвет для издержек, превышающих 5%. Базовым является красный цвет на обычном фоне.

colors.medium

Переменная medium задаёт цвет для издержек, превышающих 0,5% (по умолчанию зелёный на обычном фоне).

colors.normal

Переменная normal задаёт цвет для издержек, не относящихся к top, medium, selected (по умолчанию светло-серый на обычно фоне).

colors.selected

Переменная selected задаёт цвет и фон текущей записи в списке записей субкоманд (top, report, annotate). По умолчанию чёрный цвет на светло-сером фоне.

colors.jump_arrows

Цвета для стрелок перехода в ассемблерном коде, таких как jns, jmp, jane и т. п. По умолчанию голубой на обычном фоне.

colors.addr

Цвета для адресов в аннотации. По умолчанию пурпурный на обычном фоне.

colors.root

Цвета для заголовков в выводе субкоманд (top, report). По умолчанию белый на синем фоне.

core.*, core.proc-map-timeout

Задаёт тайм-аут (в миллисекундах, по умолчанию 500) при разборе файлов /proc/<pid>/maps. Может быть переопределен опциями --proc-map-timeout в поддерживаемых субкомандах.

tui., gtk.

Субкоманды, которые могут быть настроены (top, report и annotate). Например,

```
[tui]
        top = true
```

будет включать в TUI по умолчанию команду top. Команда будет доступна при наличии требуемых библиотек в момент сборки perf.

buildid.*, buildid.dir

Каждый исполняемый файл и библиотека общего пользования в современных дистрибутивах имеет идентификатор содержимого, который (при доступности) помещается в заголовок файла perf.data, чтобы во время анализа можно было найти информацию для преобразования символов, аннотирования кода и т. п.

Инструменты записи сохраняют ссылку или копию двоичных файлов, общих библиотек, файлов /proc/kallsyms и /proc/kcore в пользовательском файле $HOME/.debug/ для использования при анализе.

Переменная buildid.dir может применяться для изменения каталога кэширования или запрета такого кэширования (buildid.dir = /dev/null). По умолчанию используется каталог $HOME/.debug

annotate.*

Эти переменные относятся лишь к интерфейсу TUI и контролируют адреса, функции перехода, строки исходного кода в ассемблерном коде из конкретных программ.

annotate.hide_src_code

Если анализируемая программа доступна в исходном коде, эта опция управляет выводом ассемблерного кода в сопровождении исходного кода. Рассмотрим в качестве примера приведённую ниже часть программы из 4 строк. Если эта опция включена, вывод не будет сопровождаться исходным кодом.

```
        push   %rbp
        mov    %rsp,%rbp
        sub    $0x10,%rsp
        mov    (%rdi),%rdx
```

Но при выключенной (false) опции будет выводиться также исходный код (применяется по умолчанию).

```
      struct rb_node *rb_next(const struct rb_node *node)
      {
        push   %rbp
        mov    %rsp,%rbp
        sub    $0x10,%rsp
              struct rb_node *parent;

              if (RB_EMPTY_NODE(node))
        mov    (%rdi),%rdx
              return n;
```

annotate.use_offset

Можно использовать смещение от первого адреса загруженной функции. Вместо применения исходных адресов ассемблерного кода указывается разница с базовым адресом. Рассмотрим это на примере. Если базовый адрес имеет значение 0XFFFFFFFF81624d50, как показано ниже

```
ffffffff81624d50 <load0>
```

и ассемблерный код имеет абсолютный адрес

```
ffffffff816250b8:  mov    0x8(%r14),%rdi
```

то при use_offset = true выводится разница с базовым адресом (принято по умолчанию). Опция применима лишь в TUI.

```
368:  mov    0x8(%r14),%rdi
```

annotate.jump_arrows

В ассемблерном коде могут быть инструкции перехода. В зависимости от логического значения переменной jump_arrows могут выводиться стрелки, представляющие переход, как показано ниже.

```
      ┌─jmp    1333
      │ xchg   %ax,%ax
1330: │ mov    %r15,%r10
1333: └→cmp    %r15,%r14
```

Если jump_arrow = false, переход не будет показан (принято по умолчанию).

```
      ↓ jmp    1333
        xchg   %ax,%ax
1330:   mov    %r15,%r10
1333:   cmp    %r15,%r14
```

annotate.show_linenr

При выводе исходного кода значение true для этой опции задаёт вывод номеров строк, как показано ниже.

```
1628         if (type & PERF_SAMPLE_IDENTIFIER) {
     ↓ jne    508
1628                 data->id = *array;
1629                 array++;
1630         }
```

Если опция имеет значение false (принято по умолчанию), номера строк не выводятся.

```
             if (type & PERF_SAMPLE_IDENTIFIER) {
     ↓ jne    508
                     data->id = *array;
                     array++;
             }
```

annotate.show_nr_jumps

Позволяет выводить часть ассемблерного кода.

```
1382:   movb   $0x1,-0x270(%rbp)
```

При включённой опции может выводится число ветвлений, приводящих на этот адрес, как показано ниже. По умолчанию это отключено (false).

```
1 1382:   movb   $0x1,-0x270(%rbp)
```

annotate.show_total_period

Эта переменная управляет выводом числа выборок, относящихся к строке ассемблерного кода. Если переменная имеет значение true, выводятся суммарные периоды вместо процентных значений.

```
302 |       mov    %eax,%eax
```

Но при значении false (принято по умолчанию) выводится процент издержек.

```
99.93 |      mov    %eax,%eax
```

annotate.offset_level

По умолчанию установлено значение 1, при котором выводится смещение цели перехода справа от инструкции. При значении 2 будут также показаны смещения для инструкций вызова, а при значениях 3 и больше выводятся смещения для всех инструкций.

hist.*, hist.percentage

Эта переменная управляет способом расчёта издержек отфильтрованных записей и принимается во внимание только при наличии фильтра (comm, dso или имя символа). Рассмотрим пример

```
Overhead  Symbols
........  .......
 33.33%     foo
 33.33%     bar
 33.33%     baz
```

Здесь показаны исходные издержки и если отфильтровать первую запись foo, то при значении данной переменной relative издержки bar и baz примут значения 50.00%, а при значении absolute сохранится текущее значение 33.33%.

ui.*, ui.show-headers

Эта переменная управляет выводом столбцов (например, Overhead и Symbol) для команд report и top в интерфейсе TUI. При значении false имена столбцов не выводятся.

call-graph.*

Для команд top и report с опцией -g/--children эти переменные управляют графом вызовов.

call-graph.record-mode

Для команды record может быть выбран режим fp (указатель кадра), dwarf или lbr. Значение dwarf работает только при обнаружении программой perf нужной библиотеки (libunwind или свежая версия libdw). Режим lbr работает только для поддерживающих его процессоров.

call-graph.dump-size

Размер стека для дампа (по умолчанию 8192 байта). В режиме записи dwarf используется принятое по умолчанию значение, если размер не указан.

call-graph.print-type

Задаёт тип вывод graph (абсолютный граф), fractal (относительный граф), flat (плоский) или folded (свёртка). Эта переменная управляет выводом значений издержек для каждой записи цепочки вызовов. Рассмотрим пример

```
Overhead  Symbols
........  .......
  40.00%  foo
          |
          ---foo
             |
             |--50.00%--bar
             |          main
             |
              --50.00%--baz
                        main
```

Здесь используется формат fractal. Вызов foo делится поровну между bar и baz, поэтому показаны значения 50.00% для каждого (100% общих издержек foo). В режиме graph указываются абсолютные издержки foo, поэтому для bar и baz будут показаны значения 20.00%. В режиме flat выводится одна строка с линейным представлением цепочек вызовов. В режиме folded цепочки вызовов выводятся в одну строку через точку с запятой (;).

call-graph.order

Управляет порядков вывода цепочек вызовов. По умолчанию используется режим callee, где сверху указывается вызываемая функция, а далее - вызывающие в обратном порядке.

Если эта опция не установлена, а для report.children или top.children задано значение true (или эквивалентная опция в командной строке), применяется режим caller для команд perf report и perf top. Для остальных команд по умолчанию используется режим callee.

call-graph.sort-key

Цепочки вызовов с одинаковой информацией объединяются. Способ сравнения цепочек задаётся ключом сортировки, который быть функцией или адресом. По умолчанию ключ является функцией.

call-graph.threshold

При наличии множества цепочек вызовов вывод будет очень большим. Поэтому perf опускает мелкие цепочки с издержками ниже заданного этой переменной порога (по умолчанию 0.5 %). Расчёт издержек зависит от значения переменной call-graph.print-type.

call-graph.print-limit
Максимальное число строк цепочки вызовов для одного элемента гистограммы. По умолчанию число не ограничено (0).

report.*, report.sort_order
Позволяет сменить принятый по умолчанию порядок сортировки comm,dso,symbol на другой, например, sym,dso.

report.percent-limit
Похожа на call-graph.threshold, но применяется для элементов гистограммы. По умолчанию установлен порог 0. При percent-limit = 10 будут выводиться лишь записи с издержками более 10%.

report.queue-size
Задаёт максимальный размер для внутренней очереди событий (по умолчанию не ограничен - 0).

report.children
При значении этой переменной true (принято по умолчанию) команда perf report собирает цепочки вызовов дочерних функций и показывает их общие издержки вместе с издержками родительской функции (Self).

report.group
Задаёт совместный вывод информации для группы событий. Ниже приведён пример вывода со включённой опцией, где указывается один столбец на группу.

```
# group: {ref-cycles,cycles}
# ========
#
# Samples: 7K of event 'anon group { ref-cycles, cycles }'
# Event count (approx.): 6876107743
#
#         Overhead  Command      Shared Object                Symbol
# ................  .......  .................  ...................
#
    99.84%  99.76%  noploop  noploop            [.] main
     0.07%   0.00%  noploop  ld-2.15.so         [.] strcmp
     0.03%   0.00%  noploop  [kernel.kallsyms]  [k] timerqueue_del
```

top.*, top.children
То же, что report.children. При включённой опции вывод команды top включает столбец издержек Children, а также Self (по умолчанию true).

man.*, man.viewer
Эта опция может задавать программу для просмотра страниц руководства по команде help. Поддерживаются программы man, woman (с клиентом emacs) и konqueror. По умолчанию применяется man.
Новую программу просмотра руководств можно также задать с помощью man.<tool>.cmd или man.<tool>.path.

pager.*, pager.<subcommand>
Для интерфейса stdio определяет программу разбиения на страницы (pager). По умолчанию не задана.

kmem.*, kmem.default
Задаёт средство распределения, используемое при анализе, если не задана ни одна из опций --slab и --page. По умолчанию принимается slab.

record.*, record.build-id
Эта переменная может принимать значение cache, no-cache или skip. Вариант cache служит для постобработки данных и сохранения/обновления двоичной информации в кэше идентификаторов сборки (~/.debug). Этот вариант применяется по умолчанию. При выборе no-cache кэш идентификаторов сборки не обновляется, а при выборе skip не выполняется постобработки и не обновляется кэш.

diff.*, diff.order
Задаёт номер столбца для сортировки результатов. Установленное по умолчанию значение 0 обеспечивает сортировку по базовой линии, при значении 1 сортировка будет выполняться по приращению (или другому выбранному методу расчёта).

diff.compute
Задаёт метод определения различий и может принимать значения delta (по умолчанию), delta-abs, ratio и wdiff.

trace.*, trace.add_events
Позволяет добавить набор событий к указанным пользователем или применять этот набор, если события не указаны. Изначально добавляется augmented_raw_syscalls.o для активизации логики трассировки perf с просмотром содержимого указателей syscall после обычных данных точки трассировки.

trace.args_alignment
Число столбцов (терминала) для размещения аргументов (по умолчанию 70). Для strace по умолчанию используется 40. 0 отменяет выравнивание.

trace.no_inherit
Не следовать за дочерними потоками.

trace.show_arg_names
Управляет выводом имён. аргументов syscall. При отключённом выводе будет установлено trace.show_zeros.

trace.show_duration
Показывать продолжительность syscall.

trace.show_prefix
При установке значения yes будут показаны базовые префиксы строк в таблицах. По умолчанию базовые префиксы удаляются. Например, вместо MAP_SHARED будет выведено SHARED.

trace.show_timestamp
Показывать стартовую временную метку syscall.

trace.show_zeros
Не подавлять аргументы syscall со значением 0.

llvm.*, llvm.clang-path
Путь к clang. Если путь не указан, выполняется поиск по значению переменной окружения $PATH.

llvm.clang-bpf-cmd-template
Шаблон командной строки. Ниже показаны используемые по умолчанию значения. Для передачи опций применяются переменные окружения. "$CLANG_EXEC -DKERNEL $CLANG_OPTIONS $KERNEL_INC_OPTIONS \ -Wno-unused-value -Wno-pointer-sign -working-directory \ $WORKING_DIR -c $CLANG_SOURCE -target bpf -O2 -o -"

llvm.clang-opt
Опции, передаваемые clang.

llvm.kbuild-dir
Каталог kbuild. Если значение не задано, применяется /lib/modules/uname -r/build. При установке "" автоматическое определение заголовков ядра пропускается.

llvm.kbuild-opts
Опции, передаваемые make при обнаружении опций заголовков ядра.

llvm.dump-obj
Разрешает объектные файлы perf dump BPF, скомпилированные LLVM.

llvm.opts
Опции, передаваемые llc.

samples.*, samples.context
Определяет число наносекунд для отображения выборок в контексте браузера perf report.

scripts.*
Любая опция, добавляющая в меню интерактивного браузера perf script, вывод которого будет отображаться. Имя опции является именем сценария, значение задаёт командную строку сценария. Сценарий получает те же параметры, которые переданы perf, в частности -i perf.data, --cpu, --tid.

test

Команда служит для проверки работоспособности программы, а также выполняет поиск каталога, в котором находится больше всего тестов в форме сценариев. Для выполнения лишь части тестов можно указать фрагмент имён, задающий нужные тесты.

Для просмотра списка доступных тестов служит команда perf test list, которая позволяет ввести часть имени теста для поиска по подстроке.

Синтаксис

```
perf test [<options>] [{list <test-name-fragment>|[<test-name-fragments>|<test-numbers>]}]
```

Опции

-s, --skip
Пропустить тесты, указанные через запятую.

-v, --verbose
Подробный вывод.

-F, --dont-fork
Не создавать дочерних ветвлений для каждого теста, а выполнить все тесты в одном потоке.

version

Выводит версию двоичного файла perf. При указании опции --build-options выводится статус встроенных библиотек.

Синтаксис

```
perf version [--build-options]
```

Опции

--build-options
Показать статус встроенных библиотек.

Примеры использования perf

Для общего обзора - perf report --sort comm,dso.
Для выборки связанных событий - perf record -e '{cycles,instructions}:S'
Для сравнения результатов - perf diff [<old file> <new file>]
Логические опции могут обращаться с помощью no - perf report --no-children
Настройка вывода perf script - perf script -F event,ip,sym
Генерация сценария для ваших данных - perf script -g <lang>
Запись вывода perf stat - perf stat record <target workload>
Создание архива с символами для анализа на другой машине - perf archive
Поиск опций по ключевому слову - perf report -h <keyword>
использование родительского фильтра для просмотра пути конкретного вызова - perf report -p <regex>
Список событий, содержащих подстроку - perf list <keyword>
Просмотр списка сохранённых событий и атрибутов - perf evlist -v
Для нестандартного размещения файлов с символами используйте опцию --symfs <dir>
Для компактного вывода цепочки вызовов - perf report -g folded
Для просмотра отдельных выборок - perf script
Для просмотра только записей с издержками более 5% - perf report --percent-limit 5
Профилировка предсказаний ветвления - perf record -b, perf report
Просмотр ассемблерного контекста выборки record -b, perf script -F +brstackinsn --xed
Считать ветвления цепочками вызовов - perf report --branch-history
Подсчёт событий каждую секунду (1000 мсек) - perf stat -I 1000
Вывод счётчиков событий в формате CSV - perf stat -x,
Видеть отладочную информацию - perf report -s sym,srcline
Профилирование адресов памяти - perf mem record, perf mem report
События точек трассировки - perf report -s trace_fields
Запись цепочек вызовов для каждой выборки - perf record -g
Запись каждого процесса определённого пользователя - perf record -u <user>
Пропуск идентификаторов сборки при записи - perf record -B
Смена частоты выборки на 100 Гц - perf record -F 100
Вывод ассемблерных инструкций с процентами - perf annotate <symbol>
Использовать стиль Intel для ассемблера - perf annotate -M intel
Иерархический вывод результатов - perf report --hierarchy

Упорядочение по издержкам для имён файлов и строк исходного кода - perf report -s srcline
Запись данных от всех процессоров - perf record -a
Просмотр текущих конфигурационных переменных perf config --list
Показ пользовательских переопределений конфигурации - perf config --user --list
Добавление Node.js USDT[1] - perf buildid-cache --add `which node`
Просмотр событий cacheline из сделанной записи - perf c2c report
Просмотр контекста выборок - perf report --sample 10 и выбор в меню context
Разделение выборок по времени - perf report --sort time,overhead,sym

Литература

[1] The Common Trace Format, https://diamon.org/ctf/#specification

[2] AMD64 Architecture Programmer's Manual Volume 2: System Programming, 13.3 Instruction-Based Sampling, https://www.amd.com/system/files/TechDocs/24593.pdf

[3] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, https://www.intel.ru/content/www/ru/ru/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.html

[4] Static Probe Points, https://sourceware.org/gdb/onlinedocs/gdb/Static-Probe-Points.html

[5] The Linux Kernel documentation, https://www.kernel.org/doc/html/latest/.

По материалам документации ядра Linux

Николай Малых

nmalykh@protokols.ru

[1]User-Level Statically Defined Tracing - статически заданная трассировка на пользовательском уровне.