

trace-cmd

Программа для взаимодействия со встроенным трассировщиком ядра Linux [ftrace](#).

Синтаксис

```
trace-cmd COMMAND [OPTIONS]
```

Описание

Программа trace-cmd взаимодействует с трассировщиком ftrace, встроенным в ядро Linux. Взаимодействие происходит через файлы файловой системы debugfs. Параметр COMMAND задаёт операции, выполняемые trace-cmd.

Разработчиком программы является Steven Rostedt, <rostedt@goodmis.org>, исходный код доступен по ссылке <https://git.kernel.org/pub/scm/utils/trace-cmd/trace-cmd.git/>.

Команды

record

Задаёт запись трассировки в файл trace.dat на локальном диске или через сеть.

report

Читает файл trace.dat и преобразует двоичные данные в удобочитаемый текст ASCII.

profile

Запускает профилирование и считывание вывода напрямую.

hist

Выводит гистограмму событий.

stat

Показывает статус системы трассировки (ftrace).

extract

Извлекает данные из кольцевого буфера ядра и создаёт файл trace.dat.

show

Показывает содержимое буфера трассировки ядра.

options

Выводит список опций плагинов, доступных для отчёта.

start

Запускает трассировку без записи в файл trace.dat.

stop

Останавливает трассировку (прекращается лишь запись, а издержки трассировщика сохраняются).

restart

Повторно запускает остановленную ранее трассировку (воздействует лишь на запись).

reset

Отключает все трассировки, возвращая полную производительность системы (данные из буферов ядра теряются).

split

Расщепляет файл trace.dat на более мелкие файлы.

list

Выводит список доступных плагинов, событий и опций ftrace.

listen

Открывает порт для прослушивания соединений от удалённых точек трассировки.

restore

Восстанавливает файлы данных отказавшего запуска команды trace-cmd record.

stack

Запускает и показывает трассировщик стека.

check-events

Анализирует строки формата для всех событий и возвращает информацию о возможности разбора всех.

clear

Очищает буферы трассировки.

stream

Запускает трассировку и считывает вывод напрямую.

snapshot

Делает «моментальный снимок» действующей трассировки.

Опции

-h, --help

Выводит краткую справку о поддерживаемых командах.

Другие опции приведены в описаниях соответствующих команд.

record

Записывает трассировку из внутреннего трассировщика ядра Linux ftrace.

Синтаксис

```
trace-cmd record [OPTIONS] [command]
```

Описание

Команда trace-cmd record будет переводить встроенный трассировщик ядра Linux в режим записи указанных подключаемых модулей (plugin) и событий при выполнении пользовательской команды (command). Если команда для выполнения не указана, запись будет продолжаться до прерывания пользователем с помощью клавиш Ctrl-C.

При записи ftrace отслеживать события и подключаемые модули, указанные в командной строке. Это может приводить к запуску множества процессов трассировки (по одному на CPU), которые будут записываться из кольцевого буфера ядра напрямую во временные файлы. По завершении работы команды (или при нажатии Ctrl-C) все временные файлы будут объединены в один файл trace.dat, который потом может использоваться для анализа (см. report).

Опции

-p tracer

Задаёт трассировщик, который обычно не просто отслеживает события, но и выполняют другие операции. Обычно применяются трассировщики function, function_graph, preemptirqsoff, irqsoff, preemptoff и wakeup. Трассировщик должен поддерживаться работающим ядром. Список доступных трассировщиков можно посмотреть с помощью команды list.

-e event

Задаёт событие для трассировки. В ядре Linux имеется множество статических точек трассировки, сгруппированных по подсистемам, в которых можно включить все события или выбрать из них нужные. События указываются в формате subsystem:event-name. Можно просто указать подсистему без :event-name или event-name без subsystem:. Например, опция -e sched_switch будет включать событие sched_switch, а -e sched - все события подсистемы sched.

Параметр event может также включать шаблонные выражения. Т. е. *stat* будет выбирать все события (или подсистемы), в именах которых присутствует stat.

Для включения всех событий служит ключевое слово all.

-a

Каждое записываемое событие имеет файл формата, помещаемый в начало выходного файла трассировки (заголовок). Если разрешена трассировка событий, не известных trace-cmd, формат этих событий не будет записываться и отчёт trace-cmd не будет включать их. В таких случаях опция -a позволяет сохранить формат для всех событий в системе.

-T

Включает трассировку стека для каждого события. Например,

```
<idle>-0 [003] 58549.289091: sched_switch: kworker/0:1:0 [120] R ==> trace-cmd:2603 [120]
<idle>-0 [003] 58549.289092: kernel_stack: <stack trace>
=> schedule (ffffffffff814b260e)
=> cpu_idle (ffffffffff8100a38c)
=> start_secondary (ffffffffff814ab828)
```

--func-stack

Включает трассировку стека для всех функций. Отметим, что это применимо только для подключаемого модуля function и будет работать только с опцией -l для ограничения числа функций. Если трассировка функций не фильтруется и включена трассировка стека, это может «подвесить» машину.

-f filter

Задаёт фильтр для предыдущего события и должна следовать после опции -e. Опция обеспечивает фильтрацию записываемых событий на основе их содержимого. Фильтры передаются напрямую ядру, поэтому возможности фильтрации зависят от используемой версии ядра. По сути это позволяет применять нотацию языка C для решения вопроса об обработке события.

==, >=, <=, >, <, &, |, && and ||

Приведённые выше операторы обычно можно безопасно применять для сравнения полей.

--no-filter

Отключает фильтрацию потоков (thread) в trace-cmd. По умолчанию потоки фильтруются, чтобы не отслеживаться событиями. Эта опция приводит к отслеживанию потоков командой trace-cmd.

-R trigger

Задаёт триггер для предыдущего события и должна указываться после опции -e. Это позволяет добавить указанный триггер к данному событию. Чтобы включить лишь триггер, но не событие, следует поместить событие после опции -v.

Дополнительную информацию о триггерах можно найти в файле Documentation/trace/events.txt кода ядра Linux.

-v

Опция будет отключать трассировку всех событий, указанных после неё в командной строке. Это полезно при выборе подсистемы для трассировки с пропуском ненужных событий. Например, -e sched -v -e "*stat*" будет обеспечивать трассировку всех событий sched, за исключением тех, в имени которых присутствует stat.

Примечание. Опция -v была заимствована из grep, где она инвертирует последующие совпадения.

-F

Эта опция будет фильтровать лишь исполняемый файл, указанный в командной строке, а при отсутствии его - самое себя (бессмысленно). Использование опции -F позволяет трассировать лишь события, вызванные данной командой.

-P

Похожа на -F, но позволяет задать идентификатор процесса для трассировки.

-c

Используется с опцией -F (или -P, если ядро поддерживает её) для трассировки также потомков процесса.

-C clock

Устанавливает значение clock для часов трассировки. Для проверки доступности часов используйте команду trace-cmd list -C.

-o output-file

По умолчанию отчёт trace-cmd сохраняется в файле trace.dat, но эта опция позволяет задать другой файл.

-l function-name

Будет ограничивать трассировщики function и function_graph лишь указанной параметром function-name функцией. Для трассировки нескольких функций опция -l может указываться неоднократно. Возможно ограниченное применение выражений-шаблонов вида match*, когда будут трассироваться лишь функции, начинающиеся с match, и *match для функций, завершающихся match. Выражению *match* будут соответствовать функции, содержащие match.

-g function-name

Эта опция служит для плагина function_graph и будет создавать граф заданной функции. Т. е. будет трассироваться только указанная функция и все вызываемые из неё функции. Опция -g может применяться многократно.

-n function-name

Опция имеет противоположный опции -l эффект - указанная -n функция не будет трассироваться. Эта опция имеет более высокий приоритет, поэтому при указании одной функции в -n и -l эта функция не будет трассироваться.

-d

Некоторые плагины трассировки (например, трассировщик задержек) включают трассировщик function по умолчанию. Эта опция предотвращает включение трассировки функций при старте.

-D

Опция -d будет пытаться использовать опцию function-trace для запрета трассировщика функции (если это возможно), в противном случае будет по умолчанию использоваться файл /proc/sys/kernel/ftrace_enabled, но без использования для него операции touch, если опция function-trace доступна. Опция -D будет отключать (0) файл /proc/ftrace_enabled, а также опцию function-trace, если она есть.

Отметим, что это отключение относится ко всем пользователям, включая находящихся вне трассировщиков ftrace (stack_tracer, perf и т. п.).

-O option

Трассировщик ftrace имеет различные опции, которые можно включать и отключать. Добавление по перед именем опции отключает её. Например, -O no-graph-time отключит опцию graph-time.

-s interval

Процессы, создаваемые trace-cmd для записи из кольцевого буфера, нужно активизировать. Установка нулевого интервала будут активизировать процессы при каждой новой записи данных в буфер. Однако, поскольку ftrace записывает активность ядра, возврат этих процессов в спящий режим может вызывать новые события, которые снова будут будить процесс, что приведёт к добавлению в буфер избыточных данных.

Интервал задаётся в микросекундах и по умолчанию имеет значение 1000 (1 мсек). Это время, в течение которого каждый процесс будет находиться в спящем режиме, прежде чем проснуться для записи новых данных из кольцевого буфера.

-r priority

Приоритет для запуска захвата потоков. В загруженной системе потоки захвата трассировки могут быть заторможены с потерей событий. Данная опция повышает уровень приоритета до real time (FIFO). Опцию следует применять с осторожностью, поскольку она может менять поведение отслеживаемой системы.

-b size

Эта опция устанавливает размер кольцевого буфера в килобайтах. Поскольку кольцевые буферы ftrace работают на уровне CPU, заданный опцией размер выделяется в ядре для буфера каждого процессора. Опция -b 10000 на машине с 4 CPU будет создавать буферы ftrace общим размером 40 Мегабайт.

-B buffer-name

Если ядро поддерживает множество буферов, эта опция будет добавлять буфер с заданным именем. Если такой буфер уже имеется, он просто будет сброшен, а по завершении записи не будет удаляться. Если создаётся новый буфер, он будет удалён по завершении записи (если не используется опция -k или команда start).

После создания буфера все последующие добавляемые события будут связываться с ним. Если буфер не задан или событие указано до создания буфера, оно будет связано с основным (верхнего уровня) буфером.

-m size

Максимальный размер (в килобайтах) буфера, который следует держать для каждого процессора. Отметим, что в результате округления до размера страницы число может быть не вполне корректным. Кроме того, выполняется переключение между двумя буферами половинного размера, поэтому выходные данные могут не иметь в точности указанный размер даже при записи большего объёма данных.

Применение этой опции позволяет предотвратить нехватку дискового пространства для журнальных файлов.

-M cpmask

Устанавливает маску процессора cpmask для трассировки. Опция влияет лишь на последний данный экземпляр буфера. Значение маски должно быть шестнадцатеричным числом.

`trace-cmd record -p function -M c -B events13 -e all -M 5`

Если опция -M пропущена, маска не изменяется. Для трассировки всех CPU используется маска -1.

-k

По умолчанию при завершении трассировки trace-cmd сбрасывает буферы и отключает все включённые трассировки. Эта опция отменяет выключение трассировки и сброс буферов. Опция полезна для отладки trace-cmd.

Примечание. Обычно trace-cmd возвращает файл tracing_on в состояние, существовавшее до вызова. Эта опция оставит нулевой (пустой) файл.

-i

По умолчанию trace-cmd завершает работу с ошибкой, если указанного в командной строке события не существует. Эта опция позволяет игнорировать события, указанные в командной строке, но не найденные в системе.

-N host:port

Если другая машина работает в режиме trace-cmd listen, эта опция позволяет передать данные такой машине по протоколу UDP. Вместо записи в выходной файл данные передаются удаленной машине. Это очень удобно для встраиваемых систем с небольшим хранилищем или для сохранения на одной машине данных с разных систем.

Примечание. Эта опция не поддерживается с плагины трассировщиков задержки wakeup, wakeup_rt, irqsoff, preemptoff и preemptirqsoff

-t

Эта опция применяется вместе с -N, когда нужно передавать данные на удалённую машину по протоколу TCP вместо UDP. Хотя TCP работает медленней UDP, этот протокол может потребоваться в ненадёжной сети, если объем передаваемых данных невелик, но нужна гарантированная доставка.

-q | --quiet

Подавляет обычный вывод программы (за исключением ошибок), отображая лишь данные отслеживаемого приложения.

--date

Опция задаёт trace-cmd запись временных меток в буфер трассировки по завершении записи. Опция будет отображать timestamp в gettimeofday для вывода системного времени при чтении созданного файла trace.dat.

--max-graph-depth depth

Задаёт максимальную глубину трассировки `function_graph` в функции. При значении 1 будет указываться лишь пользовательский вход в ядро без каких-либо функций, вызываемых в ядре. По умолчанию применяется неограниченная глубина (0).

--module module

Задаёт фильтрацию имени модуля в трассировке. Это эквивалентно добавлению `:mod:module` после всех других функций, которые будут фильтроваться. Если других фильтров функцией не указано, будут фильтроваться все функции.

--module snd эквивалентно -l :mod:snd;

--module snd -l "*"jack*" эквивалентно -l "*"jack*:mod:snd";

--module snd -n "*" эквивалентно -n :mod:snd.

--profile

С опцией --profile программа `trace-cmd` будет включать трассировку, которая может использоваться с командой `trace-cmd report --profile`. Если опция -p не задана и глубина графа функции поддерживается ядром, трассировщик `function_graph` будет включён с глубиной 1 (показывать лишь пользовательский вход в ядро). Будут также включены различные трассировочные точки с трассировкой стека, так что отчёт может показать, где задачи были заблокированы дольше всего.

Дополнительная информация и примеры представлены в описании `profile`.

-H event-hooks

Добавляет пользовательское сопоставление событий для связывания вместе пары событий. При использовании вместе с опцией --profile параметр сохраняется и будет применяться также командой `trace-cmd report --profile`. Т. е. команды

```
trace-cmd record -H hrtimer_expire_entry,hrtimer/hrtimer_expire_exit,hrtimer,sp
trace-cmd report --profile
```

будут профилировать время `hrtimer_expire_entry` и `hrtimer_expire_ext`.

Формат приведён в описании `profile`.

-S

(только --profile) Включает лишь трассировщик или события, указанные в командной строке. С этой опцией трассировщик `function_graph`, равно как и другие события (такие как `sched_switch`) не включаются, пока не указаны явно в командной строке (например, -p function -e sched_switch -e sched_wakeup).

--ts-offset offset

Добавляет смещение для временных меток в файле `trace.dat`. Смещение задаётся в необработанных единицах, т. е. при записи меток в наносекундах смещение также будет в наносекундах, даже если для отображения применяются миллисекунды.

--stderr

Задаёт вывод в `stderr` вместо `stdout` без изменения выходного потока отслеживаемой команды. Это полезно в тех случаях, когда хочется видеть вывод команды без вывода `trace-cmd`.

Примеры

Базовый вариант трассировки всех событий

```
# trace-cmd record -e all ls > /dev/null
# trace-cmd report
trace-cmd-13541 [003] 106260.693809: filemap_fault: address=0x128122 offset=0x0ce
trace-cmd-13543 [001] 106260.693809: kmalloc: call_site=81128dd4 ptr=0xffff88003dd83800 bytes_req=768
bytes_alloc=1024 gfp_flags=GFP_KERNEL|GFP_ZERO
ls-13545 [002] 106260.693809: kfree: call_site=810a7abb ptr=0x0
ls-13545 [002] 106260.693818: sys_exit_write: 0x1
```

Для трассировки с отслеживанием `sched_switch`

```
# trace-cmd record -p function -e sched_switch ls > /dev/null
# trace-cmd report
-13587 [002] 106467.860310: function: hrtick_start_fair <-- pick next task fair
ls-13587 [002] 106467.860313: sched_switch: - prev_comm=trace-cmd prev_pid=13587 prev_prio=120
prev_state=R ==> next_comm=trace-cmd next_pid=13583 next_prio=120
trace-cmd-13585 [001] 106467.860314: function: native_set_pte at <-- do fault
trace-cmd-13586 [003] 106467.860314: function: up_read <-- do_page_fault
ls-13587 [002] 106467.860317: function: _phys_addr <-- schedule
trace-cmd-13585 [001] 106467.860318: function: raw_spin_unlock <-- do fault
ls-13587 [002] 106467.860320: function: native_load_sp0 <-- switch to
trace-cmd-13586 [003] 106467.860322: function: down_read_trylock <-- do_page_fault
```

Ниже показан хороший способ обнаружить прерывания с наибольшей задержкой.

```
# trace-cmd record -p function_graph -e irq_handler_entry -l do_IRQ sleep 10
# trace-cmd report
<idle>-0 [000] 157412.933969: funcgraph entry: | do_IRQ() {
<idle>-0 [000] 157412.933974: irq_handler entry: | irq=48 name=eth0
<idle>-0 [000] 157412.934004: funcgraph_exit: | + 36.358 us | }
<idle>-0 [000] 157413.895004: funcgraph entry: | do_IRQ() {
<idle>-0 [000] 157413.895011: irq_handler entry: | irq=48 name=eth0
<idle>-0 [000] 157413.895026: funcgraph_exit: | + 24.014 us | }
<idle>-0 [000] 157415.891762: funcgraph entry: | do_IRQ() {
<idle>-0 [000] 157415.891769: irq_handler entry: | irq=48 name=eth0
<idle>-0 [000] 157415.891784: funcgraph_exit: | + 22.928 us | }
<idle>-0 [000] 157415.934869: funcgraph entry: | do_IRQ() {
<idle>-0 [000] 157415.934874: irq_handler entry: | irq=48 name=eth0
<idle>-0 [000] 157415.934906: funcgraph_exit: | + 37.512 us | }
<idle>-0 [000] 157417.888373: funcgraph entry: | do_IRQ() {
<idle>-0 [000] 157417.888381: irq_handler entry: | irq=48 name=eth0
<idle>-0 [000] 157417.888398: funcgraph_exit: | + 25.943 us | }
```

Ниже приведён пример записи профиля.

```
# trace-cmd record --profile sleep 1
# trace-cmd report --profile --comm sleep
task: sleep-21611
Event: sched_switch:R (1) Total: 99442 Avg: 99442 Max: 99442 Min:99442
<stack> 1 total:99442 min:99442 max:99442 avg:99442
=> ftrace_raw_event sched_switch (0xffffffff8105f812)
=> _schedule_ (0xffffffff8150810a)
=> preempt_schedule (0xffffffff8150842e)
=> preempt_schedule (0xffffffff81273354)
=> cpu_stop_queue_work (0xffffffff810b03c5)
=> stop_one_cpu (0xffffffff810b063b)
=> sched_exec (0xffffffff8106136d)
=> do_execve_common.isra.27 (0xffffffff81148c89)
=> do_execve (0xffffffff811490b0)
=> Sys_execve (0xffffffff811492c4)
=> return_to_handler (0xffffffff8150e3c8)
=> stub_execve (0xffffffff8150c699)
Event: sched_switch:S (1) Total: 1000506680 Avg: 1000506680 Max: 1000506680 Min:1000506680
```

```

<stack> 1 total:1000506680 min:1000506680 max:1000506680 avg=1000506680
=> ftrace raw event sched switch (0xffffffff8105f812)
=> _schedule (0xffffffff8150810a)
=> _schedule (0xffffffff815084b8)
=> do_nanosleep (0xffffffff8150b22c)
=> hrTimer_nanosleep (0xffffffff8108d647)
=> Sys_nanosleep (0xffffffff8108d72c)
=> return_to_handler (0xffffffff8150e3c8)
=> tracesys_phase2 (0xffffffff8150c304)
Event: sched_wakeup:21611 (1) Total: 30326 Avg: 30326 Max: 30326 Min:30326
<stack> 1 total:30326 min:30326 max:30326 avg=30326
=> ftrace raw event sched_wakeup_template (0xffffffff8105f653)
=> ttwu_do_wakeup (0xffffffff810606eb)
=> ttwu_do_activate.constprop.124 (0xffffffff810607c8)
=> try_to_wake_up (0xffffffff8106340a)

```

report

Показывает текст ASCII с результатами трассировки, записанной ранее в файл.

Синтаксис

```
trace-cmd report [OPTIONS] [input-file]
```

Описание

Команда trace-cmd report выводит понятный человеку отчёт о трассировке, выполненной по команде trace-cmd record.

Опции

-i input-file

По умолчанию trace-cmd будет читать файл отчёта. trace.dat, но эта опция позволяет указать иной файл параметром input-file. Отметим, что входной файл можно также задать в конце командной строки (без опции -i).

-e

Эта опция позволяет увидеть тип файла отчёта. Команда trace-cmd достаточно эффективна и может читать файлы big endian на машинах little endian и наоборот.

-f

Обеспечивает вывод списка функций, записанных в файл.

-P

Обеспечивает вывод списка данных trace_printk(). Необработанные данные трассировки показывают статические указатели в ядре. Эти данные должны сохраняться в файле трассировки.

-E

Обеспечивает вывод списка возможных событий в файле (это не обязательно список реально записанных событий).

--events

Обеспечивает вывод списка форматов событий, сохранённых в файле трассировки.

--event regex

Обеспечивает вывод событий, соответствующих регулярному выражению regex. При включении двоеточия символы перед ним служат для сопоставления системы, а символы после двоеточия - для сопоставления событий.

```
trace-cmd report --event sys:read
```

Приведённая выше команда будет соответствовать лишь системам, содержащим в имени sys, и событиям, содержащим read.

```
trace-cmd report --event read
```

Эта команда будет соответствовать всем событиям и системам, содержащим read в имени.

--check-events

Задаёт анализ строк формата событий в файле трассировки и возврат информации о возможности их корректного анализа. Опция ведёт к загрузке плагинов, если не задана опция -N.

-t

Задаёт вывод временных меток, которые обычно записываются в файл данных в наносекундах, однако по умолчанию выводятся лишь миллисекунды. Эта опция позволяет увидеть временные метки полностью.

-F

Добавляет фильтр отображаемых событий. Формат фильтра показан ниже.

```

<events> ':' <filter>
<events> = SYSTEM '/' EVENT | SYSTEM | EVENT | <events> ',' <events>
<filter> = EVENT_FIELD <op> <value> | <filter> '&&' <filter> | <filter> '||' <filter> | '(' <filter> ')' | '!' <filter>
<op> = '==' | '!=' | '>=' | '<=' | '>' | '<' | '&' | '|' | '^' | '+' | '-' | '*' | '/' | '%'
<value> = NUM | STRING | EVENT_FIELD

```

SYSTEM указывает имя системы для фильтра. Если поле EVENT опущено, фильтр применяется ко всем событиям SYSTEM. Если используется лишь одна строка без символа / для разделения SYSTEM и EVENT, фильтр будет применяться ко всем системам и событиям, соответствующим заданной строке.

Пробельные символы игнорируются, поэтому строки "sched:next_pid==123" и "sched : next_pid == 123" эквивалентны.

Строка STRING указывается в одинарных или двойных кавычках (одинаковых с обеих сторон), пробельные символы внутри кавычек принимаются во внимание.

В качестве SYSTEM и/или EVENT могут также указываться регулярные выражения, соответствующие regcomp.

EVENT_FIELD указывает имя фильтруемого поля в событии. Если событие не включает EVENT_FIELD, эта часть выражения считается ложной (false).

```
-F 'sched : bogus == 1 || common_pid == 2'
```

Выражение bogus == 1 всегда будет иметь значение FALSE, поскольку ни в одном событии нет поля bogus, но common_pid == 2 все равно будет проверяться, поскольку все события имеют поле common_pid. Все события sched, трассируемые процессом с PID = 2, будут показаны.

Отметим, что EVENT_FIELD является именем поля, показываемым в формате события (как при выводе с опцией *--events*), а не в выводе. Если вывод показывает ID:foo, но поле, к которому относится foo, называется name в формате события, в качестве фильтра нужно применять name. То же самое относится к значениям. Если отображаемое значение конвертировано в строку символов, фильтр проверяет исходное, а не отображаемое значение. Например, для фильтрации всех задач, которые работали в контексте switch служит выражение

```
-F 'sched/sched_switch : prev_state==0'
```

Хотя вывод показывает R, фильтр prev_stat=="R" не будет работать.

Примечание. Можно также указать COMM в качестве EVENT_FIELD. При этом для сравнения будет использоваться имя задачи (или of comm) в записи. Например, для фильтрации всех задач trace-cmd можно применить выражение

```
-F '.*:COMM != "trace-cmd"'
```

-I

Отключает вывод событий с установленным флагом задержки HARDIRQ. Это будет фильтровать большинство событий из контекста прерывания. Отметим, что это не поможет отфильтровать функции трассировки функций в контексте прерывания, которые были вызваны до установки флага ядра "in interrupt".

-S

Отключает вывод событий с установленным флагом задержки SOFTIRQ. Это будет фильтровать большинство событий из контекста программного прерывания.

-v

Включает фильтры, указанные в последующих опциях -F, для совпадающих событий.

```
-v -F 'sched/sched_switch : prev_state == 0'
```

Приведённое выражение исключит все события sched_switch с prev_state = 0. Удаление -v приведёт лишь к выводу этих событий.

-T

Проверка фильтров опции -F. После обработки строки фильтра для каждого события будет выводиться результирующий фильтр. Это полезно при использовании фильтра для нескольких событий, когда поле может присутствовать во всех событиях может применяться также для гарантии отсутствия некорректно введённых имён событий, которые будут просто игнорироваться. Опция -T не принимается во внимание при отсутствии опции -F.

-V

Задаёт вывод загружаемых плагинов.

-L

Отменяет загрузку плагинов системного уровня, загружая лишь локальные (т. е. расположенные в каталоге ~/.trace-cmd/plugins).

-N

Отменяет загрузку всех плагинов.

-n event-re

Заставляет все события, соответствующие опции, игнорировать все зарегистрированные (плагином) обработчики для вывода события, используя взамен обычный формат. Параметр event-re является регулярным выражением, соответствующим regexp.

--profile

С этой опцией команда trace-cmd report будет сначала обрабатывать все события, а затем выводить формат, показывающий, где задачи тратили своё время в ядре, а также где они были дольше всего заблокированы и где была наибольшая задержка пробуждения.

Дополнительная информация и примеры приведены в описании команды profile.

-G

Устанавливает событие прерывания (программного или аппаратного) как глобальное (связанное с CPU, а не задачей). Работает только в опции --profile.

-H event-hooks

Добавляет пользовательское сопоставление событий для связывания вместе пары событий.

Формат приведён в описании profile.

-R

Задаёт вывод событий в необработанном (raw) виде, т. е. форматирование для печати будет игнорироваться.

-r event-re

Задаёт вывод всех соответствующих опции событий в необработанном формате. Параметр event-re является регулярным выражением, соответствующим regexp.

-I

Добавляет формат «вывода задержки». Информация о прерываниях и программные прерывания отключаются, устанавливается флаг need_resched, записывается счётчик вытеснения и большая блокировка ядра (big kernel lock) с каждым событием. Однако по умолчанию эта информация не отображается. Данная опция задаёт 6-символьное отображение отмеченных данных. Если одно из полей равно нулю или недоступно (N/A), выводится '\.'

```
<idle>-0 0d.h1. 106467.859747: function: ktime_get <-- tick_check_idle
```

Поле 0d.h1. указывает эту информацию. Точка никогда не бывает первым символом, который указывает на каком процессоре была записана трассировка (CPU 0). Буква d указывает запрет прерывания, h говорит о вызове внутри обработчика прерывания, 1 указывает, что для запрета вытеснения (preempt_count) было установлено значение 1. Две точки являются флагом need_resched и счётчиком блокировки ядра. Если флаг need_resched установлен, вместо точки будет показан символ N.

-w

Если включены события sched_switch и sched_wakeup, эта опция будет включать вывод задержки между первым вызовом задачи и запланированным временем.

-q

Отключает некритичные предупреждения.

-O

Передаёт опции в загружаемые плагины trace-cmd.

```
-O plugin:var=value
```

Параметры plugin: и =value являются необязательными. Значения логических параметров можно пропускать. Если параметр plugin: не включён, будут установлены любые переменные, соответствующие всем плагинам.

```
-O fgraph:tailprint --stat
```

Если в файле трассировки записана финальная статистика (выводится в конце записи), для её извлечения может служить опция --stat.

--uname

Если в файл трассировки при выполнении записывались данные uname, опция будет извлекать эту информацию.

--version

Если в файл трассировки при выполнении записывалась версия исполняемой программы, опция обеспечивает её вывод.

```
--ts-offset offset
```

Добавляет (вычитает при отрицательном значении) смещение ко всем временным меткам предыдущего файла данных, заданного опцией -i. Это полезно для сортируемого слияния множества файлов трассировки с разными временными метками, например, при выполнении трассировки на виртуальной машине одновременно с другой трассировкой на хосте. Если временная метка хоста опережает гостевую на 1000 единиц, можно воспользоваться командой

```
trace-cmd report -i host.dat --ts-offset -1000 -i guest.dat
```

Это будет вычитать 1000 единиц времени для всех событий на хосте при слиянии с событиями из файла guest.dat. Отметим, что единицы относятся к необработанным данным, записанным в файл. Если единицей является наносекунда, вычитаться или добавляться будут наносекунды даже при отображении времени в микросекундах.

--ts2secs HZ

Задаёт преобразование текущего источника времени во второй (с наносекундным разрешением) выход. При использовании часов типа x86-tsc и известной частоте время можно преобразовать в секунды путём указания частоты.

Эта опция влияет на любой файл трассировки, переданный с обработкой -i. Если опция расположена до всех опций -i, значение становится используемым по умолчанию преобразованием для всех других файлов трассировки. При наличии другой опции --ts2secs после -i trace.dat, эта опция будет переопределять принятое по умолчанию значение.

Например, при частоте 3.4 ГГц и командах

```
trace-cmd record -p function -C x86-tsc
trace-cmd report --ts2ns 3400000000
```

отчёт будет преобразовывать временные метки тактов в читаемые значения секунд. По умолчанию будет применяться микросекундное разрешение, если не задана опция -t.

Значение --ts-offset должно оставаться в единицах необработанных меток даже при наличии этой опции, поскольку сдвиг также будет преобразован.

--ts-diff

Показывает временную разницу между событиями (в скобках после временной метки).

Примеры

При использовании файла trace.dat, созданного командой

```
# trace-cmd record -p function -e all sleep 5
```

отчёт по умолчанию будет иметь вид

```
# trace-cmd report
trace-cmd-16129 [002] 158126.498411: function: mutex_unlock_slowpath <-- mutex_unlock
trace-cmd-16131 [000] 158126.498411: kmem_cache_alloc: call_site=811223c5 ptr=0xfffff88003ecf2b40 bytes_req=272
bytes_alloc=320 gfp_flags=GFP_KERNEL|GFP_ZERO
trace-cmd-16130 [003] 158126.498411: function: do_splice_to <-- sys_splice
sleep-16133 [001] 158126.498412: function: inotify_inode_queue_event <-- vfs_write
trace-cmd-16129 [002] 158126.498420: lock_release: 0xfffff88003f1fa4f8 &sb->s_type->i_mutex_key
trace-cmd-16131 [000] 158126.498421: function: security_file_alloc <-- get_empty_filp
sleep-16133 [001] 158126.498422: function: fsnotify_parent <-- vfs_write
trace-cmd-16130 [003] 158126.498422: function: rw_verify_area <-- do_splice_to
trace-cmd-16131 [000] 158126.498424: function: cap_file_alloc_security <-- security_file_alloc
trace-cmd-16129 [002] 158126.498425: function: syscall_trace_leave <-- int_check_syscall_exit_work
sleep-16133 [001] 158126.498426: function: inotify_dentry_parent_queue_event <-- vfs_write
trace-cmd-16130 [003] 158126.498426: function: security_file_permission <-- rw_verify_area
trace-cmd-16129 [002] 158126.498428: function: audit_syscall_exit <-- syscall_trace_leave
[...]
```

Для просмотра без трассировки функций служит команда

```
# trace-cmd report -v -F 'function'
trace-cmd-16131 [000] 158126.498411: kmem_cache_alloc: call_site=811223c5 ptr=0xfffff88003ecf2b40 bytes_req=272
bytes_alloc=320 gfp_flags=GFP_KERNEL|GFP_ZERO
trace-cmd-16129 [002] 158126.498420: lock_release: 0xfffff88003f1fa4f8 &sb->s_type->i_mutex_key
trace-cmd-16130 [003] 158126.498436: lock_acquire: 0xfffff88003f16b6f78 read_all_cpu_access_lock
trace-cmd-16131 [000] 158126.498438: lock_acquire: 0xfffff88003df5b520 read &fs->lock
trace-cmd-16129 [002] 158126.498446: kfree: call_site=810a7abb ptr=0x0
trace-cmd-16130 [003] 158126.498448: lock_acquire: 0xfffff880002250a80 &per_cpu(cpu_access_lock, cpu)
trace-cmd-16129 [002] 158126.498450: sys_exit_splice: 0xfffffff5
trace-cmd-16131 [000] 158126.498454: lock_release: 0xfffff88003df5b520 &fs->lock
sleep-16133 [001] 158126.498456: kfree: call_site=810a7abb ptr=0x0
sleep-16133 [001] 158126.498460: sys_exit_write: 0x1
trace-cmd-16130 [003] 158126.498462: kmalloc: call_site=810bf95b ptr=0xfffff88003dedc040 bytes_req=24
bytes_alloc=32 gfp_flags=GFP_KERNEL|GFP_ZERO
```

Для просмотра лишь вызовов kmalloc с размером больше 1000 служит команда

```
# trace-cmd report -F 'kmalloc: bytes req > 1000'
<idle>->0 [000] 158128.126641: kmalloc: call_site=81330635 ptr=0xfffff88003c2fd000 bytes_req=2096
bytes_alloc=4096 gfp_flags=GFP_ATOMIC
```

Для просмотра пробуждений и sched_switch, оставивших предыдущую задачу в рабочем состоянии, служит команда

```
# trace-cmd report -F 'sched: prev_state == 0 || (success == 1)'
trace-cmd-16132 [001] 158126.499951: sched_wakeup: comm=trace-cmd pid=16129 prio=120 success=1 target_cpu=002
trace-cmd-16132 [002] 158126.500401: sched_switch: prev_comm=trace-cmd prev_pid=16132 prev_prio=120 prev_state=R ==>
next_comm=trace-cmd next_pid=16129 next_prio=120
<idle>->0 [003] 158126.500585: sched_wakeup: comm=trace-cmd pid=16130 prio=120 success=1 target_cpu=003
<idle>->0 [003] 158126.501241: sched_switch: prev_comm=swapper prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=trace-cmd next_pid=16130 next_prio=120
trace-cmd-16132 [000] 158126.502475: sched_wakeup: comm=trace-cmd pid=16131 prio=120 success=1 target_cpu=000
trace-cmd-16131 [002] 158126.506516: sched_wakeup: comm=trace-cmd pid=16129 prio=120 success=1 target_cpu=002 <idle>->0
[003] 158126.550110: sched_switch: prev_comm=swapper prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=trace-cmd
next_pid=16130 next_prio=120
trace-cmd-16131 [003] 158126.570243: sched_wakeup: comm=trace-cmd pid=16129 prio=120 success=1 target_cpu=003 trace-cmd-
16130 [002] 158126.618202: sched_switch: prev_comm=trace-cmd prev_pid=16130 prev_prio=120 prev_state=R ==> next_comm=yum-
updatesd next_pid=3088 next_prio=1 20
trace-cmd-16129 [003] 158126.622379: sched_wakeup: comm=trace-cmd pid=16131 prio=120 success=1 target_cpu=003 trace-cmd-
16129 [000] 158126.649287: sched_wakeup: comm=trace-cmd pid=16131 prio=120 success=1 target_cpu=000
```

Приведённый выше пример требует некоторых пояснений. Фильтр задаёт подсистему sched, включающую события sched_switch и sched_wakeup. Любое событие, не имеющее поля формата prev_state или success, будет давать для выражения значение FALSE (отсутствие совпадения). Использование || будет задавать проверку prev_state для sched_switch и success для sched_wakeup.

```
# trace-cmd report -w -F 'sched_switch, sched_wakeup.*'
[...]
trace-cmd-16130 [003] 158131.580616: sched_wakeup: comm=trace-cmd pid=16131 prio=120 success=1 target_cpu=003
trace-cmd-16129 [000] 158131.581502: sched_switch: prev_comm=trace-cmd prev_pid=16129 prev_prio=120 prev_state=S ==>
next_comm=trace-cmd next_pid=16131 next_prio=120 Latency: 885.901 usecs
trace-cmd-16131 [000] 158131.582414: sched_wakeup: comm=trace-cmd pid=16129 prio=120 success=1 target_cpu=000
trace-cmd-16132 [001] 158131.583219: sched_switch: prev_comm=trace-cmd prev_pid=16132 prev_prio=120 prev_state=S ==>
next_comm=trace-cmd next_pid=16129 next_prio=120 Latency: 804.809 usecs
sleep-16133 [002] 158131.584121: sched_wakeup: comm=trace-cmd pid=16120 prio=120 success=1 target_cpu=002
```

```

trace-cmd-16129 [001] 158131.584128: sched wakeup: comm=trace-cmd pid=16132 prio=120 success=1 target_cpu=001
sleep-16133 [002] 158131.584275: sched switch: prev comm=sleep prev_pid=16133 prev_prio=120 prev_state=R ==>
next_comm=trace-cmd next_pid=16120 next_prio=120 Latency: 153.915 usecs
trace-cmd-16130 [003] 158131.585284: sched switch: prev comm=trace-cmd prev_pid=16130 prev_prio=120 prev_state=S ==>
next_comm=trace-cmd next_pid=16132 next_prio=120 Latency: 1155.677 usecs

```

Average wakeup latency: 26626.656 usecs

Приведённая выше трассировка показывает задержку пробуждения задач. Событие sched_switch указывает отдельную задержку после записи информации о событии. В конце отчёта, указана средняя задержка пробуждения.

```

# trace-cmd report -w -F 'sched switch, sched wakeup.*: prio < 100 || next_prio < 100'
<idle>-0 [003] 158131.516753: sched wakeup: comm=ksoftirqd/3 pid=13 prio=49 success=1 target_cpu=003
<idle>-0 [003] 158131.516855: sched switch: prev comm=swapper prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=ksoftirqd/3 next_pid=13 next_prio=49 Latency: 101.244 usecs
<idle>-0 [003] 158131.533781: sched wakeup: comm=ksoftirqd/3 pid=13 prio=49 success=1 target_cpu=003
<idle>-0 [003] 158131.533897: sched switch: prev comm=swapper prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=ksoftirqd/3 next_pid=13 next_prio=49 Latency: 115.608 usecs
<idle>-0 [003] 158131.569730: sched wakeup: comm=ksoftirqd/3 pid=13 prio=49 success=1 target_cpu=003
<idle>-0 [003] 158131.569851: sched switch: prev comm=swapper prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=ksoftirqd/3 next_pid=13 next_prio=49 Latency: 121.024 usecs

```

Average wakeup latency: 110.021 usecs

Приведённый выше вариант показывает лишь пробуждения и переключатели контекста задач Real Time. Используемый в ядре приоритет рrio имеет значение 0 в качестве максимального приоритета. Это значение 0 эквивалентно приоритету 99 в пользовательском пространстве, а пользовательский приоритет 98 эквивалентен приоритету 1 в ядре. Значения рrio меньше 100 представляют задачи в реальном масштабе времени (Real Time).

Ниже приведён пример профиля.

```

# trace-cmd record --profile sleep 1
# trace-cmd report --profile --comm sleep
task: sleep-21611
Event: sched switch:R (1) Total: 99442 Avg: 99442 Max: 99442 Min:99442
<stack> 1 total:99442 min:99442 max:99442 avg=99442
=> ftrace_raw_event sched switch (0xffffffff8105f812)
=> schedule (0xffffffff8150810a)
=> preempt_schedule (0xffffffff8150842e)
=> preempt_schedule (0xffffffff81273354)
=> cpu_stop_queue_work (0xffffffff810b03c5)
=> stop_one_cpu (0xffffffff810b063b)
=> sched_exec (0xffffffff8106136d)
=> do_execve_common.isra.27 (0xffffffff81148c89)
=> do_execve (0xffffffff811490b0)
=> Sys_execve (0xffffffff811492c4)
=> return_to_handler (0xffffffff8150e3c8)
=> stub_execve (0xffffffff8150c699)
Event: sched switch:S (1) Total: 1000506680 Avg: 1000506680 Max: 1000506680 Min:1000506680
<stack> 1 total:1000506680 min:1000506680 max:1000506680 avg=1000506680
=> ftrace_raw_event sched switch (0xffffffff8105f812)
=> schedule (0xffffffff8150810a)
=> schedule (0xffffffff815084b8)
=> do_nanosleep (0xffffffff8150b22c)
=> hrtimer_nanosleep (0xffffffff8108d647)
=> Sys_nanosleep (0xffffffff8108d72c)
=> return_to_handler (0xffffffff8150e3c8)
=> tracesys_phase2 (0xffffffff8150c304)
Event: sched wakeup:21611 (1) Total: 30326 Avg: 30326 Max: 30326 Min:30326
<stack> 1 total:30326 min:30326 max:30326 avg=30326
=> ftrace_raw_event sched wakeup template (0xffffffff8105f653)
=> ttwu_do_wakeup (0xffffffff810606eb)
=> ttwu_do_activate.constprop.124 (0xffffffff810607c8)
=> try_to_wake_up (0xffffffff8106340a)

```

profile

Профилировка работы задачи.

Синтаксис

```
trace-cmd profile [OPTIONS] [command]
```

Описание

Команда trace-cmd profile будет запускать трассировку, подобно команде trace-cmd record --profile, за исключением того, что запись в файл не будет осуществляться и события будут считываться по мере возникновения с одновременным их учётом. По завершении трассировки будет выводиться отчёт как по команде trace-cmd report --profile. Иными словами, команда profile работает подобно trace-cmd record --profile и trace-cmd report --profile без промежуточной записи данных на диск.

Преимущество использования команды profile заключается в том, что она избавляет от записи промежуточных файлов на диск, которые при длительной трассировке могут занимать очень много места.

Эта команда включает некоторые события, а также трассировщик function_graph с глубиной 1 (если поддерживает ядро). Это позволяет видеть вход задач в ядро и выход из него, а также продолжительность обработки в ядре.

Для запрета вызова function_graph служит опция -p, включающая другой трассировщик. Для отключения всех трассировщиков служит -p por.

Время при профилировании указывается в наносекундах.

Опции

Команда использует те же опции, что и trace-cmd record --profile.

-p tracer

Задаёт плагин трассировки для использования вместо function_graph с глубиной 1. Для отключения всех трассировщиков служит -p por.

-S

Включает лишь трассировщик или события, указанные в командной строке. С этой опцией трассировщик function_graph, равно как и другие события (такие как sched_switch) не включаются, пока не указаны явно в командной строке (например, -p function -e sched_switch -e sched_wakeup).

-G

Устанавливает события прерываний (программных и аппаратных) как глобальные (связанные с CPU, а не задачей).

-o file

Задаёт запись профиля в файл file. Это заменяет опцию --stderr.

-H event-hooks

Добавляет пользовательское сопоставление событий для связывания вместе пары событий в формате

```
[<start_system>:<start_event>,<start_match>|,<start_pid>] / [<end_system>:<end_event>,<end_match>|,<end_pid>]
```

Поле start_system:start_event (start_system не обязательно) указывает событие, с которого начинается отсчёт времени. Поле start_match в стартовом событии служит для сопоставления с полем end_match в финальном событии. Параметр start_pid является необязательным, поскольку сопоставления присоединяются к задачам, которые запускают события. Если для поиска этой задачи следует применять другое поле, оно указывается start_pid. Параметр end_system:end_event указывает событие, на котором завершается отсчёт времени (end_system не обязательно). Поле end_match в финальном событии сопоставляется с полем start_match в стартовом событии. Необязательное поле может включать приведённые ниже флаги (без учёта регистра).

p - два события привязаны к одному CPU (начало и завершение всегда происходят на одном CPU).

s - событию следует иметь отслеживаемый стек (включается трассировка стека для стартового события).

g - событие является глобальным (не связано с задачей). Параметр start_pid не применим с этим флагом.

--stderr

Перенаправляет вывод профиля в stderr, не изменяя вывода трассируемой команды. Это позволяет видеть вывод команды, сохраняя вывод профиля в другом файле.

Примеры

```
# trace-cmd profile -F sleep 1
[...]
```

```
task: sleep-1121
Event: sched_switch:R (2) Total: 234559 Avg: 117279 Max: 129886 Min:104673
+ ftrace_raw_event sched_switch (0xffffffff8109f310)
  100% (2) time:234559 max:129886 min:104673 avg:117279
    schedule (0xffffffff816c1e81)
    preempt_schedule (0xffffffff816c236e)
    preempt_schedule (0xffffffff81351a59)
      + unmap_single_vma (0xffffffff81198c05)
        55% (1) time:129886 max:129886 min:0 avg:129886
        stop_one_cpu (0xffffffff8110909a)
        sched_exec (0xffffffff810a119b)
        do_execveat_common.isra.31 (0xffffffff811de528)
        do_execve (0xffffffff811dea8c)
        Sys_execve (0xffffffff811ded1e)
        return_to_handler (0xffffffff816c8458)
        stub_execve (0xffffffff816c6929)
      + unmap_single_vma (0xffffffff81198c05)
        45% (1) time:104673 max:104673 min:0 avg:104673
        unmap_vmas (0xffffffff81199174)
        exit_mmap (0xffffffff811a1f5b)
        mmap (0xffffffff8107699a)
        flush_old_exec (0xffffffff811ddb75)
        load_elf_binary (0xffffffff812287df)
        search_binary_handler (0xffffffff811dd3e0)
        do_execveat_common.isra.31 (0xffffffff811de8bd)
        do_execve (0xffffffff811dea8c)
        Sys_execve (0xffffffff811ded1e)
        return_to_handler (0xffffffff816c8458)
        stub_execve (0xffffffff816c6929)
Event: sched_switch:S (1) Total: 1000513242 Avg: 1000513242 Max: 1000513242 Min:1000513242
+ ftrace_raw_event sched_switch (0xffffffff8109f310)
  100% (1) time:1000513242 max:1000513242 min:0 avg:1000513242
    schedule (0xffffffff816c1e81)
    schedule (0xffffffff816c23b9)
    do_nanosleep (0xffffffff816c4f1c)
    hrtimer_nanosleep (0xffffffff810dcd86)
    Sys_nanosleep (0xffffffff810dcea6)
    return_to_handler (0xffffffff816c8458)
    tracesys_phase2 (0xffffffff816c65b0)
Event: sched_wakeup:1121 (1) Total: 43405 Avg: 43405 Max: 43405 Min:43405
+ ftrace_raw_event sched_wakeup_template (0xffffffff8109d960)
  100% (1) time:43405 max:43405 min:0 avg:43405
    ttwu_do_wakeup (0xffffffff810a01a2)
    ttwu_do_activate.constprop.122 (0xffffffff810a0236)
    try_to_wake_up (0xffffffff810a3ec3)
    wake_up_process (0xffffffff810a4057)
    hrtimer_wakeup (0xffffffff810db772)
    run_hrtimer (0xffffffff810dbd91)
    hrtimer_interrupt (0xffffffff810dc6b7)
    local_apic_timer_interrupt (0xffffffff810363e7)
    smp_trace_apic_timer_interrupt (0xffffffff816c8c6a)
    trace_apic_timer_interrupt (0xffffffff816c725a)
    finish_task_switch (0xffffffff8109c3a4)
    schedule (0xffffffff816c1e01)
    schedule (0xffffffff816c23b9)
    ring_buffer_wait (0xffffffff811323a3)
    wait_on_pipe (0xffffffff81133d93)
    tracing_buffers_splice_read (0xffffffff811350b0)
    do_splice_to (0xffffffff8120476f)
    Sys_splice (0xffffffff81206c1f)
    tracesys_phase2 (0xffffffff816c65b0)
Event: func: sys_nanosleep() (1) Total: 1000598016 Avg: 1000598016 Max: 1000598016 Min:1000598016
Event: func: sys_munmap() (1) Total: 14300 Avg: 14300 Max: 14300 Min:14300
Event: func: sys_arch_prctl() (1) Total: 571 Avg: 571 Max: 571 Min:571
Event: func: sys_mprotect() (4) Total: 14382 Avg: 3595 Max: 7196 Min:2190
Event: func: Sys_read() (1) Total: 2640 Avg: 2640 Max: 2640 Min:2640
Event: func: sys_close() (5) Total: 4001 Avg: 800 Max: 1252 Min:414
Event: func: sys_newstat() (3) Total: 11684 Avg: 3894 Max: 10206 Min:636
Event: func: Sys_open() (3) Total: 23615 Avg: 7871 Max: 10535 Min:4743
Event: func: sys_access() (1) Total: 5924 Avg: 5924 Max: 5924 Min:5924
Event: func: Sys_mmap() (8) Total: 39153 Avg: 4894 Max: 12354 Min:1518
Event: func: smp_trace_apic_timer_interrupt() (1) Total: 10298 Avg: 10298 Max: 10298 Min:10298
Event: func: Sys_brk() (4) Total: 2407 Avg: 601 Max: 1564 Min:206
Event: func: do_notify_resume() (2) Total: 4095 Avg: 2047 Max: 2521 Min:1574
Event: func: sys_execve() (5) Total: 1625251 Avg: 325050 Max: 1605698 Min:3570
+ ftrace_raw_event sched_wakeup_template (0xffffffff8109d960)
  100% (1) time:1605698 max:1605698 min:0 avg:1605698
    ttwu_do_wakeup (0xffffffff810a01a2)
    ttwu_do_activate.constprop.122 (0xffffffff810a0236)
```

```
try to wake up (0xffffffff810a3ec3)
wake_up_proc (0xffffffff810a4057)
cpu_stop_queue_work (0xffffffff81108df8)
stop_one_cpu (0xffffffff8110909a)
sched_exec (0xffffffff810a119b)
do_execveat_common.isra.31 (0xffffffff811de528)
do_execve (0xffffffff811dea8c)
Sys_execve (0xffffffff811ded1e)
return to handler (0xffffffff816c8458)
stub_execve (0xffffffff816c6929)
stub_execve (0xffffffff816c6929)

Event: func: syscall_trace_enter_phase2() (38) Total: 21544 Avg: 566 Max: 1066 Min:329
Event: func: syscall_trace_enter_phase1() (38) Total: 9202 Avg: 242 Max: 376 Min:150
Event: func: __do_page_fault() (53) Total: 257672 Avg: 4861 Max: 27745 Min:458
|
+ ftrace_raw_event sched_wakeup_template (0xffffffff8109d960)
100% (1) time:27745 max:27745 min:0 avg:27745
ttwu_do_wakeup (0xffffffff810a01a2)
ttwu_do_activate.constprop.122 (0xffffffff810a0236)
try to wake up (0xffffffff810a3ec3)
default_wake_function (0xffffffff810a4002)
autoremove_wake_function (0xffffffff810b50fd)
__wake_up_common (0xffffffff810b4958)
__wake_up (0xffffffff810b4cb8)
PB_wake_up_waiters (0xffffffff8112f126)
irq_work_run_list (0xffffffff81157d0f)
irq_work_run (0xffffffff81157d5e)
smp_trace_irq_work_interrupt (0xffffffff810082fc)
trace_irq_work_interrupt (0xffffffff816c7aaa)
return to handler (0xffffffff816c8458)
trace_do_page_fault (0xffffffff810478b2)
trace_page_fault (0xffffffff816c7dd2)

Event: func: syscall_trace_leave() (38) Total: 26145 Avg: 688 Max: 1264 Min:381
Event: func: __sb_end_write() (1) Total: 373 Avg: 373 Max: 373 Min:373
Event: func: fsnotify() (1) Total: 598 Avg: 598 Max: 598 Min:598
Event: func: __fsnotify_parent() (1) Total: 286 Avg: 286 Max: 286 Min:286
Event: func: mutex_unlock() (2) Total: 39636 Avg: 19818 Max: 39413 Min:223
Event: func: smp_trace_irq_work_interrupt() (6) Total: 236459 Avg: 39409 Max: 100671 Min:634
|
+ ftrace_raw_event sched_wakeup_template (0xffffffff8109d960)
100% (4) time:234348 max:100671 min:38745 avg:58587
ttwu_do_wakeup (0xffffffff810a01a2)
ttwu_do_activate.constprop.122 (0xffffffff810a0236)
try to wake up (0xffffffff810a3ec3)
default_wake_function (0xffffffff810a4002)
autoremove_wake_function (0xffffffff810b50fd)
__wake_up_common (0xffffffff810b4958)
__wake_up (0xffffffff810b4cb8)
PB_wake_up_waiters (0xffffffff8112f126)
irq_work_run_list (0xffffffff81157d0f)
irq_work_run (0xffffffff81157d5e)
smp_trace_irq_work_interrupt (0xffffffff810082fc)
return to handler (0xffffffff816c8458)
trace_irq_work_interrupt (0xffffffff816c7aaa)
|
+ ftrace_return_to_handler (0xffffffff81140840)
84% (73) time:197396 max:100671 min:38745 avg:65798
|
| return to handler (0xffffffff816c846d)
| trace_page_fault (0xffffffff816c7dd2)
|
+ ftrace_return_to_handler (0xffffffff81140840)
16% (71) time:36952 max:36952 min:0 avg:36952
ftrace_graph_caller (0xffffffff816c8428)
mutex_unlock (0xffffffff816c3f75)
rb_simple_write (0xffffffff81133142)
vfs_write (0xffffffff811d7727)
Sys_write (0xffffffff811d7acf)
tracesys_phase2 (0xffffffff816c65b0)

Event: sys_enter:35 (1) Total: 1000599765 Avg: 1000599765 Max: 1000599765 Min:1000599765
Event: sys_enter:11 (1) Total: 55025 Avg: 55025 Max: 55025 Min:55025
Event: sys_enter:158 (1) Total: 1584 Avg: 1584 Max: 1584 Min:1584
Event: sys_enter:10 (4) Total: 18359 Avg: 4589 Max: 8764 Min:2933
Event: sys_enter:0 (1) Total: 4223 Avg: 4223 Max: 4223 Min:4223
Event: sys_enter:3 (5) Total: 9948 Avg: 1989 Max: 2606 Min:1203
Event: sys_enter:5 (3) Total: 15530 Avg: 5176 Max: 11840 Min:1405
Event: sys_enter:2 (3) Total: 28002 Avg: 9334 Max: 12035 Min:5656
Event: sys_enter:21 (1) Total: 7814 Avg: 7814 Max: 7814 Min:7814
Event: sys_enter:9 (8) Total: 49583 Avg: 6197 Max: 14137 Min:2362
Event: sys_enter:12 (4) Total: 108493 Avg: 27123 Max: 104079 Min:922
Event: sys_enter:59 (5) Total: 1631608 Avg: 326321 Max: 1607529 Min:4563
Event: page_fault_user:0x398d86b630 (1)
Event: page_fault_user:0x398d844de0 (1)
Event: page_fault_user:0x398d8d9020 (1)
Event: page_fault_user:0x1d37008 (1)
Event: page_fault_user:0x7f0b89e91074 (1)
Event: page_fault_user:0x7f0b89d98ad0 (1)
Event: page_fault_user:0x7f0b89ec8950 (1)
Event: page_fault_user:0x7f0b89d83644 (1)
Event: page_fault_user:0x7f0b89d622a8 (1)
Event: page_fault_user:0x7f0b89d5a560 (1)
Event: page_fault_user:0x7f0b89d34010 (1)
Event: page_fault_user:0x1d36008 (1)
Event: page_fault_user:0x398d900510 (1)
Event: page_fault_user:0x398dbb3ae8 (1)
Event: page_fault_user:0x398d87f490 (1)
Event: page_fault_user:0x398d8eb660 (1)
Event: page_fault_user:0x398d8bd730 (1)
Event: page_fault_user:0x398d9625d9 (1)
Event: page_fault_user:0x398d931810 (1)
Event: page_fault_user:0x398dbb7114 (1)
Event: page_fault_user:0x398d837610 (1)
Event: page_fault_user:0x398d89e860 (1)
Event: page_fault_user:0x398d8f23b0 (1)
Event: page_fault_user:0x398dbb4510 (1)
Event: page_fault_user:0x398dbad6f0 (1)
Event: page_fault_user:0x398dbb1018 (1)
Event: page_fault_user:0x398d977b37 (1)
Event: page_fault_user:0x398d92eb60 (1)
Event: page_fault_user:0x398d8abff0 (1)
Event: page_fault_user:0x398dbb0d30 (1)
Event: page_fault_user:0x398dbb6c24 (1)
Event: page_fault_user:0x398d821c50 (1)
Event: page_fault_user:0x398dbb6c20 (1)
Event: page_fault_user:0x398d886350 (1)
Event: page_fault_user:0x7f0b90125000 (1)
Event: page_fault_user:0x7f0b90124740 (1)
Event: page_fault_user:0x7f0b90126000 (1)
Event: page_fault_user:0x398d816230 (1)
Event: page_fault_user:0x398d8002b8 (1)
Event: page_fault_user:0x398dbb0b40 (1)
Event: page_fault_user:0x398dbb2880 (1)
Event: page_fault_user:0x7f0b90141cc6 (1)
```

```

Event: page_fault_user:0x7f0b9013b85c (1)
Event: page_fault_user:0x7f0b90127000 (1)
Event: page_fault_user:0x606e70 (1)
Event: page_fault_user:0x7f0b90144010 (1)
Event: page_fault_user:0x7fffc31b038 (1)
Event: page_fault_user:0x606da8 (1)
Event: page_fault_user:0x400040 (1)
Event: page_fault_user:0x398d222218 (1)
Event: page_fault_user:0x398d015120 (1)
Event: page_fault_user:0x398d220ce8 (1)
Event: page_fault_user:0x398d220b80 (1)
Event: page_fault_user:0x7fffc32fcff8 (1)
Event: page_fault_user:0x398d001590 (1)
Event: page_fault_user:0x398d838490 (1)
Event: softirq_raise:RCU (3) Total: 252931 Avg: 84310 Max: 243288 Min:4639
Event: softirq_raise:SCHED (2) Total: 241249 Avg: 120624 Max: 239076 Min:2173
|
+ ftrace raw event sched wakeup template (0xffffffff8109d960)
100% (1) time:239076 max:239076 min:0 avg:239076
ttwu do wakeup (0xffffffff810a01a2)
ttwu do activate.constprop.122 (0xffffffff810a0236)
try to wake up (0xffffffff810a3ec3)
default wake function (0xffffffff810a4002)
autoremove wake function (0xffffffff810b50fd)
wake_up_common (0xffffffff810b4958)
wake_up (0xffffffff810b4cb8)
zb wake up waiters (0xffffffff8112f126)
irq_work_run_list (0xffffffff81157d0f)
irq_work_run (0xffffffff81157d5e)
smp_trace_irq_work_interrupt (0xffffffff810082fc)
trace_irq_work_interrupt (0xffffffff816c7aaa)
irq_exit (0xffffffff8107dd66)
smp_trace_apic_timer_interrupt (0xffffffff816c8c7a)
trace_apic_timer_interrupt (0xffffffff816c725a)
prepare_ftrace_return (0xffffffff8103d4fd)
ftrace_graph_caller (0xffffffff816c8428)
mem_cgroup_begin_page_stat (0xffffffff811cfd25)
page_remove_rmap (0xffffffff811a4fc5)
stub_execve (0xffffffff816c6929)
unmap_single_vma (0xffffffff81198b1c)
unmap_vmas (0xffffffff81199174)
exit_mmap (0xffffffff811a1f5b)
mmap (0xffffffff8107699a)
flush_old_exec (0xffffffff811ddb75)
load_elf_binary (0xffffffff812287df)
search_binary_handler (0xffffffff811dd3e0)
do_execveat_common.isra.31 (0xffffffff811de8bd)
do_execve (0xffffffff811dea8c)
Sys_execve (0xffffffff811ded1e)
return_to_handler (0xffffffff816c8458)

Event: softirq_raise:HI (3) Total: 72472 Avg: 24157 Max: 64186 Min:3430
Event: softirq_entry:RCU (2) Total: 3191 Avg: 1595 Max: 1788 Min:1403
|
+ ftrace raw event sched wakeup template (0xffffffff8109d960)
100% (1) time:1788 max:1788 min:0 avg:1788
ttwu do wakeup (0xffffffff810a01a2)
ttwu do activate.constprop.122 (0xffffffff810a0236)
try to wake up (0xffffffff810a3ec3)
default wake function (0xffffffff810a4002)
autoremove wake function (0xffffffff810b50fd)
wake_up_common (0xffffffff810b4958)
wake_up (0xffffffff810b4cb8)
zb wake up waiters (0xffffffff8112f126)
irq_work_run_list (0xffffffff81157d0f)
irq_work_run (0xffffffff81157d5e)
smp_trace_irq_work_interrupt (0xffffffff810082fc)
trace_irq_work_interrupt (0xffffffff816c7aaa)
irq_work_queue (0xffffffff81157e95)
ring_buffer_unlock_commit (0xffffffff8113039f)
buffer_unlock_commit (0xffffffff811367d5)
trace_buffer_unlock_commit (0xffffffff811376a2)
ftrace_event_buffer_commit (0xffffffff81146d5f)
ftrace_raw_event_sched_process_exec (0xffffffff8109c511)
do_execveat_common.isra.31 (0xffffffff811de9a3)
do_execve (0xffffffff811dea8c)
Sys_execve (0xffffffff811ded1e)
return_to_handler (0xffffffff816c8458)
stub_execve (0xffffffff816c6929)

Event: softirq_entry:SCHED (2) Total: 2289 Avg: 1144 Max: 1350 Min:939
Event: softirq_entry:HI (3) Total: 180146 Avg: 60048 Max: 178969 Min:499
|
+ ftrace raw event sched wakeup template (0xffffffff8109d960)
100% (1) time:178969 max:178969 min:0 avg:178969
ttwu do wakeup (0xffffffff810a01a2)
ttwu do activate.constprop.122 (0xffffffff810a0236)
try to wake up (0xffffffff810a3ec3)
wake_up_process (0xffffffff810a4057)
wake_up_worker (0xffffffff8108de74)
insert_work (0xffffffff8108fca6)
queue_work (0xffffffff8108fe12)
delayed_work_timer_fn (0xffffffff81090088)
call_timer_fn (0xffffffff810d8f89)
run_timer_softirq (0xffffffff810da8a1)
do_softirq (0xffffffff8107d8fa)
irq_exit (0xffffffff8107dd66)
smp_trace_apic_timer_interrupt (0xffffffff816c8c7a)
trace_apic_timer_interrupt (0xffffffff816c725a)
prepare_ftrace_return (0xffffffff8103d4fd)
ftrace_graph_caller (0xffffffff816c8428)
mem_cgroup_begin_page_stat (0xffffffff811cfd25)
page_remove_rmap (0xffffffff811a4fc5)
stub_execve (0xffffffff816c6929)
unmap_single_vma (0xffffffff81198b1c)
unmap_vmas (0xffffffff81199174)
exit_mmap (0xffffffff811a1f5b)
mmap (0xffffffff8107699a)
flush_old_exec (0xffffffff811ddb75)
load_elf_binary (0xffffffff812287df)
search_binary_handler (0xffffffff811dd3e0)
do_execveat_common.isra.31 (0xffffffff811de8bd)
do_execve (0xffffffff811dea8c)
Sys_execve (0xffffffff811ded1e)
return_to_handler (0xffffffff816c8458)

```

В приведённом выше примере используется опция -F для отслеживания спящих задач, которая фильтрует события, относящиеся к сну. Отметим, что для отслеживания ветвлений нужно также указать опцию -s.

Другие задачи тоже будут включены в профиль, если события указывают более одной задачи (подобно sched_switch и sched_wakeup - prev_pid и next_pid для sched_switch, common_pid и pid для sched_wakeup).

Трассировки стека присоединяются к событиям, с которыми они связаны.

Рассмотрим приведённый ниже вывод.

```
Event: sched_switch:R (2) Total: 234559 Avg: 117279 Max: 129886 Min:104673
```

Это показывает, что задача была вытеснена (находится в состоянии R). Задача была вытеснена дважды в целом на 234559 нсек со средним временем вытеснения 117279 нсек, максимальным 128886 нсек и минимальным 104673 нсек.

Дерево ниже показывает точки вытеснения.

```

|
+ ftrace_raw_event sched switch (0xffffffff8109f310)
  100% (2) time:234559 max:129886 min:104673 avg:117279
    _schedule (0xffffffff816c1e81)
    preempt_schedule (0xffffffff816c236e)
    preempt_schedule (0xffffffff81351a59)
  + unmap_single_vma (0xffffffff81198c05)
    55% (1) time:129886 max:129886 min:0 avg:129886
    stop_one_cpu (0xffffffff8110909a)
    sched_exec (0xffffffff810a119b)
    do_execveat_common.isra.31 (0xffffffff811de528)
    do_execve (0xffffffff811dea8c)
    Sys_execve (0xffffffff811ded1e)
    return_to_handler (0xffffffff816c8458)
    stub_execve (0xffffffff816c6929)
  + unmap_single_vma (0xffffffff81198c05)
    45% (1) time:104673 max:104673 min:0 avg:104673
    unmap_vmas (0xffffffff81199174)
    exit_mmap (0xffffffff811a1f5b)
    mmpu_t (0xffffffff8107699a)
    flush_old_exec (0xffffffff811ddb75)
    load_elf_binary (0xffffffff812287df)
    search_binary_handler (0xffffffff811dd3e0)
    do_execveat_common.isra.31 (0xffffffff811de8bd)
    do_execve (0xffffffff811dea8c)
    Sys_execve (0xffffffff811ded1e)
    return_to_handler (0xffffffff816c8458)
    stub_execve (0xffffffff816c6929)

```

```
Event: sched_switch:S (1) Total: 1000513242 Avg: 1000513242 Max: 1000513242 Min:10005132
```

Это показывает, что задача была запланирована в состоянии INTERRUPTIBLE (прерываемая) один раз на суммарное время 1000513242 нсек (~1 сек.), что соответствует sleep 1.

После запланированных событий выводятся функциональные события. По умолчанию используется трассировщик function_graph, если установка глубины поддерживается ядром. Устанавливается глубина 1, при которой будет отслеживаться лишь первая функция входа в ядро. Будет записано также время выполнения в ядре.

```

Event: func: sys_nanosleep() (1) Total: 1000598016 Avg: 1000598016 Max: 1000598016 Min:1000598016
Event: func: sys_munmap() (1) Total: 14300 Avg: 14300 Max: 14300 Min:14300
Event: func: sys_arch_prctl() (1) Total: 571 Avg: 571 Max: 571 Min:571
Event: func: sys_mprotect() (4) Total: 14382 Avg: 3595 Max: 7196 Min:2190
Event: func: Sys_read() (1) Total: 2640 Avg: 2640 Max: 2640 Min:2640
Event: func: sys_close() (5) Total: 4001 Avg: 800 Max: 1252 Min:414
Event: func: sys_newstat() (3) Total: 11684 Avg: 3894 Max: 10206 Min:636
Event: func: Sys_open() (3) Total: 23615 Avg: 7871 Max: 10535 Min:4743
Event: func: sys_access() (1) Total: 5924 Avg: 5924 Max: 5924 Min:5924
Event: func: Sys_mmap() (8) Total: 39153 Avg: 4894 Max: 12354 Min:1518
Event: func: smp_trace_apic_timer_interrupt() (1) Total: 10298 Avg: 10298 Max: 10298 Min:10298
Event: func: Sys_brk() (4) Total: 2407 Avg: 601 Max: 1564 Min:206
Event: func: do_notify_resume() (2) Total: 4095 Avg: 2047 Max: 2521 Min:1574
Event: func: sys_execve() (5) Total: 1625251 Avg: 325050 Max: 1605698 Min:3570

```

Count of times the event was hit is always in parenthesis (5).

Трассировка function_graph может вызывать очень большие издержки, поскольку она продолжает срабатывать для всех (а не только отслеживаемых) функций. Для ограничения числа системных вызовов (не прерываний) можно использовать опции:

```
-l 'sys *' -l 'Sys *'
```

Для полного отключения function_graph служит опция

```
-p nop
```

Для трассировки функций (отметим, что будет записываться не время, а лишь число вызовов функции) служит опция

```
-p function
```

Показанные ниже функции являются записываемыми событиями.

```

Event: sys_enter:35 (1) Total: 1000599765 Avg: 1000599765 Max: 1000599765 Min:1000599765
Event: sys_enter:11 (1) Total: 55025 Avg: 55025 Max: 55025 Min:55025
Event: sys_enter:158 (1) Total: 1584 Avg: 1584 Max: 1584 Min:1584
Event: sys_enter:10 (4) Total: 18359 Avg: 4589 Max: 8764 Min:2933
Event: sys_enter:0 (1) Total: 4223 Avg: 4223 Max: 4223 Min:4223
Event: sys_enter:3 (5) Total: 9948 Avg: 1989 Max: 2606 Min:1203
Event: sys_enter:5 (3) Total: 15530 Avg: 5176 Max: 11840 Min:1405
Event: sys_enter:2 (3) Total: 28002 Avg: 9334 Max: 12035 Min:5656
Event: sys_enter:21 (1) Total: 7814 Avg: 7814 Max: 7814 Min:7814
Event: sys_enter:9 (8) Total: 49583 Avg: 6197 Max: 14137 Min:2362
Event: sys_enter:12 (4) Total: 108493 Avg: 27123 Max: 104079 Min:922
Event: sys_enter:59 (5) Total: 1631608 Avg: 326321 Max: 1607529 Min:4563

```

Это необработанные события системных вызовов с указанием идентификатора вызова после sys_enter:. Например, 59 - это execve. Почему эта функция вызывалась 5 раз? Рассмотрение strace показывает

```

execve("/usr/lib64/ccache/sleep", ["sleep", "1"], [/* 27 vars */] <unfinished ...>
<... execve resumed> ) = -1 ENOENT (No such file or directory)
execve("/usr/local/sbin/sleep", ["sleep", "1"], [/* 27 vars */] <unfinished ...>
<... execve resumed> ) = -1 ENOENT (No such file or directory)
execve("/usr/local/bin/sleep", ["sleep", "1"], [/* 27 vars */] <unfinished ...>
<... execve resumed> ) = -1 ENOENT (No such file or directory)
execve("/usr/sbin/sleep", ["sleep", "1"], [/* 27 vars */] <unfinished ...>
<... execve resumed> ) = -1 ENOENT (No such file or directory)
execve("/usr/bin/sleep", ["sleep", "1"], [/* 27 vars */] <unfinished ...>
<... execve resumed> ) = 0

```

Видно попытку выполнить команду sleep для каждого пути в \$PATH, пока не будет найден исполняемый файл.

События page_fault_user показывают, какой адрес в пользовательском пространстве принимает отказ страницы.

```

Event: softirq_raise:RCU (3) Total: 252931 Avg: 84310 Max: 243288 Min:4639
Event: softirq_raise:SCHED (2) Total: 241249 Avg: 120624 Max: 239076 Min:2173
|
+ ftrace_raw_event sched wakeup template (0xffffffff8109d960)
  100% (1) time:239076 max:239076 min:0 avg:239076
    ttwu_do_wakeup (0xffffffff810a01a2)
    ttwu_do_activate.constprop.122 (0xffffffff810a0236)

```

```

try to wake up (0xffffffff810a3ec3)
default wake function (0xffffffff810a4002)
autoremove wake function (0xffffffff810b50fd)
wake_up_common (0xffffffff810b4958)
wake_up (0xffffffff810b4cb8)
__wake_up_waiters (0xffffffff8112f126)
irq_work_run_list (0xffffffff81157d0f)
irq_work_run (0xffffffff81157d5e)
smp_trace_irq_work_interrupt (0xffffffff810082fc)
trace_irq_work_interrupt (0xffffffff816c7aaa)
irq_exit (0xffffffff8107dd66)

```

Время событий `softirq_raise` измеряет интервал от поднятия `softirq` до момента завершения прерывания.

Время `softirq_entry` показывает продолжительность выполнения `softirq`.

Трассировка стека для `softirq` (и возможно других событий) используется в тех случаях, когда стек присоединён к событию. Это может происходить, если профиль запускает больше стеков, чем просто события `sched`, или события отбрасываются и стекируются.

Для полного контроля трассировки служит опция `-S`, по которой `trace-cmd` не будет включать трассировщики событий и `function_graph`, показывая лишь события, указанные в командной строке.

Если нужно видеть лишь время `kmalloc` и место записи, опция `-S` и включение трассировки стека и `function_graph` лишь для нужной функции будут давать профиль лишь для этой функции.

```

# trace-cmd profile -S -p function_graph -l '*kmalloc*' -l '*kmalloc*:stacktrace' sleep 1
task: sshd-11786
Event: func: __kmalloc_reserve.isra.59() (2) Total: 149684 Avg: 74842 Max: 75598 Min: 74086
+
+  alloc_skb (0xffffffff815a8917)
+  67% (2) time:149684 max:75598 min:74086 avg:74842
+    kmalloc_node_track_caller (0xffffffff811c6635)
+    __kmalloc_reserve.isra.59 (0xffffffff815a84ac)
+    Return_to_handler (0xffffffff816c8458)
+    sk_stream_alloc_skb (0xffffffff81604ea1)
+    tcp_sendmsg (0xffffffff8160592c)
+    inet_sendmsg (0xffffffff8162fed1)
+    sock_aio_write (0xffffffff8159f9fc)
+    do_sync_write (0xffffffff811d694a)
+    vfs_write (0xffffffff811d7825)
+    Sys_write (0xffffffff811d7adf)
+    system_call_fastpath (0xffffffff816c63d2)
+
+  alloc_skb (0xffffffff815a8917)
+  33% (1) time:74086 max:74086 min:74086 avg:74086
+    alloc_skb (0xffffffff815a8917)
+    sk_stream_alloc_skb (0xffffffff81604ea1)
+    tcp_sendmsg (0xffffffff8160592c)
+    inet_sendmsg (0xffffffff8162fed1)
+    sock_aio_write (0xffffffff8159f9fc)
+    do_sync_write (0xffffffff811d694a)
+    vfs_write (0xffffffff811d7825)
+    Sys_write (0xffffffff811d7adf)
+    system_call_fastpath (0xffffffff816c63d2)
+
[...]
```

Для наблюдения вывода команды с сохранением вывода профиля в файл служит опция `--stderr` и перенаправление стандартного вывода ошибок в файл

```
# trace-cmd profile --stderr cyclicttest -p 80 -n -tl 2> profile.out
```

или просто -o

```
# trace-cmd profile -o profile.out cyclicttest -p 80 -n -tl
```

hist

Показывает гистограмму событий, записанных в файл трассировки.

Синтаксис

```
trace-cmd hist [OPTIONS] [input-file]
```

Описание

Команда `trace-cmd hist` выводит гистограмму событий из файла трассировки. Вместо показа событий в упорядоченном виде создаётся гистограмма, которая может отображаться по задачам или для всех задач с выводом наиболее общих событий в начале. Команда использует трассировщик функций и стеки вызовов для построения гистограммы событий.

Опции

-i input-file

По умолчанию `trace-cmd hist` будет считывать данные из файла `trace.dat`, но с помощью этой опции можно задать другой файл (`input-file`). Отметим, что входной файл может быть также указан последним параметром командной строки.

-P

Задаёт компактирование всех событий и отображение вызовов с игнорированием задач и разных PID. Вместо указания имён задач все цепочки группируются и выводятся как `<all pids>`.

stat

Показывает статус системы трассировки (`ftrace`).

Синтаксис

```
trace-cmd stat
```

Описание

Команда `trace-cmd stat` выводит статус системы трассировки (`ftrace`).

Tracer - если один из трассировщиков (таких как `function_graph`) активен, в остальных случаях не выводится ничего.

Events - список событий, которые разрешены.

Event filters - показывает любые фильтры, установленные для любых событий.

Function filters - показывает любые фильтры, установленные для любых трассировщиков функций.

Graph functions - показывает любые функции, которые трассировщику function_graph следует отображать.

Buffers - показывает размер буфера трассировки. По умолчанию буферы трассировки в процессе использования находятся в сжатом формате и отображение буфера не представляется.

Trace clock - если трассировочные часы отличаются от принятых по умолчанию локальных часов (local), они будут указаны.

Trace CPU mask - если не все доступные CPU попадают в маску трассировки, указывается эта маска.

Trace max latency - указывает значение максимальной задержки трассировки, если оно отлично от 0.

Kprobes - показывает любые kprobe, определённые для трассировки.

Uprobes - показывает любые uprobe, определённые для трассировки.

extract

Извлекает данные трассировщика Linux ftrace.

Синтаксис

```
trace-cmd extract [OPTIONS]
```

Описание

Команда trace-cmd extract обычно применяется после trace-cmd-start и trace-cmd-stop, а также может применяться после запуска трассировщика ftrace вручную через псевдофайловую систему.

Команда extract создаёт файл trace.dat, который может использоваться trace-cmd report для считывания и анализа данных. Исходные данные для создания файла trace.dat извлекаются из внутреннего кольцевого буфера ядра.

Опции

-p plugin

Хотя команда extract не запускает никакой трассировки, некоторые плагины требуют чтения вывода в формате ASCII. К ним относятся трассировщики задержки, поскольку они используют отдельный внутренний буфер. Опция нужна лишь для плагинов wakeup, wakeup-rt, irqsoff, preemptoff и preemptirqsoff.

Без этой опции команда extract будет извлекать внутренние буферы ftrace.

-o option

Если извлекается трассировка задержки и применяется опция -p, некоторые опции ftrace могут менять формат. Данная опция будет менять их до извлечения. Просмотреть опции можно с помощью команды trace-cmd list. Для включения опции указывается её имя, для отключения имя указывается с префиксом no. Например, noprint-parent будет отключать опцию print-parent, которая выводит родительскую функцию при отображении функционального события.

-o outfile

По умолчанию команда extract создаёт файл trace.dat, но с помощью этой опции можно задать иной файл для вывода.

-s

Задаёт извлечение данных из буфера «моментальных снимков» (snapshot), если это поддерживается ядром.

--date

То же самое, что опция trace-cmd record --date, но с отключением всей трассировки в программе извлечения. Т. е. в конце извлечения данных выполняется что-то вроде trace-cmd reset.

-B buffer-name

Если ядро поддерживает множество буферов, эта опция позволяет извлечь данные из нужного буфера. Опцию можно включать в нескольких экземплярах для указания разных буферов ядра. При указании этой опции экземпляр верхнего уровня не используется, если не задана также опция -t.

-a

Задаёт извлечение данных из всех экземпляров буферов. При указании этой опции экземпляр верхнего уровня не используется, если не задана также опция -t.

-t

Задаёт извлечение данных из буфера верхнего уровня. Без опций -B или -a это обеспечивает принятое по умолчанию поведение, но при указании -B или -a данные из буфера верхнего уровня извлекается вместе с данными из указанных этими опциями буферов.

show

Показывает содержимое буфера трассировки ядра (ftrace).

Синтаксис

```
trace-cmd show [OPTIONS]
```

Описание

Команда trace-cmd show выводит содержимое одного из файлов трассировки (ftrace) ядра Linux - trace, snapshot или trace_pipe. Это эквивалентно использованию команд вида

```
cat /sys/kernel/debug/tracing/trace
```

Опции

-p

trace-cmd

Задаёт вывод содержимого файла `trace_pipe` вместо выводимого по умолчанию `trace`, который является статическим (т. е. по завершении трассировки содержимое файла `trace` остаётся неизменным).

Чтение файла `trace_pipe` является поглощающим и данные не будут повторяться при следующем чтении. Файл при этом блокируется. Если в файле нет данных, `trace-cmd show` будет останавливаться и ждать их появления.

-s

Задаёт вывод содержимого файла `snapshot` вместо выводимого по умолчанию `trace`. Снимки делаются записывающим приложением и ядро будет переключаться между текущим активным буфером `snapshot` и следующим буфером. Без переключения файл `snapshot` является статическим. Считывание файла не поглощает данные из него.

-c cpu

Задаёт чтение файла `trace` лишь для указанного CPU.

-f

Выводит полный путь к отображаемому файлу.

-B buf

Если был создан экземпляр буфера, опция `-B` обеспечивает доступ к файлам, связанным с этим буфером.

--tracing_on

Указывает, включена ли трассировка на данном экземпляре.

--current_tracer

Показывает, что служит текущим трассировщиком.

--buffer_size

Показывает текущий размер буфера (на процессор)

--buffer_total_size

Показывает общий размер всех буферов.

--ftrace_filter

Показывает установленные фильтры функций.

--ftrace_notrace

Показывает, какие функции отключены установленными фильтрами.

--ftrace_pid

Показывает PID трассировщиков функций (если они есть).

--graph_function

Показывает функции, которые будут отображены.

--graph_notrace

Показывает функции, которые не будут отображены.

--cpumask

Показывает маску CPU, для которых будет выполняться трассировка.

options

Выводит список доступных опций для плагинов `trace-cmd`.

Синтаксис

```
trace-cmd options
```

Описание

Команда `trace-cmd options` будет проверять плагины `trace-cmd`, используемые командой `trace-cmd report`, и выводить их список.

start

Запускает трассировщик `ftrace` без записи в файл.

Синтаксис

```
trace-cmd start [OPTIONS]
```

Описание

Команда `trace-cmd start` включает всю трассировку `ftrace`, как это делает `trace-cmd record(1)`, но без запуска потоков для создания файла `trace.dat`. Это полезно, когда нужно просто включить `ftrace` и интересна лишь трассировка после того, как произошло некоторое событие и до момента остановки. Затем трассировку можно считать напрямую из псевдофайловой системы `ftrace` или извлечь с помощью команды `trace-cmd extract`.

Опции

Опции этой команды совпадают с опциями `trace-cmd record`, за исключением относящихся к записи (`-s`, `-o`, `-F`, `-N`, `-t`).

stop

Команда останавливает запись трассировщика ядра Linux `ftrace` в кольцевой буфер

Синтаксис

```
trace-cmd stop
```

Описание

Команда `trace-cmd stop` служит дополнением к `trace-cmd-start` и будет отключать запись `ftrace` в кольцевой буфер, не отменяя издержек, которые могут быть связаны с трассировкой. Отключается лишь обновление кольцевого буфера, но трассировка `ftrace` может продолжаться, создавая издержки.

После остановки команда `trace-cmd extract` позволяет прочитать данные из кольцевого буфера и создать файл `trace.dat`. Возможно также прямое считывание данных из псевдофайловой системы.

Для полного отключения трассировки с целью устранения связанных с ней издержек служит команда `trace-cmd reset`, но при сбросе записанные данные будут потеряны.

Опции

-B buffer-name

Если ядро поддерживает множество буферов, эта опция позволяет остановить трассировку для определённого буфера, не воздействуя на остальные. Опцию можно указать несколько раз для разных буферов. При указании этой опции экземпляр верхнего уровня не останавливается, если не была задана также опция `-t`.

-a

Останавливает трассировку всех имеющихся экземпляров буферов. При указании этой опции экземпляр верхнего уровня не останавливается, если не была задана также опция `-t`.

-t

Останавливает буфер верхнего уровня. Без опции `-B` или `-a` эта опция обеспечивает принятое по умолчанию поведение, но при наличии `-B` или `-a`, будет останавливаться также буфер верхнего уровня.

reset

Отключает все трассировки `ftrace`, восстанавливая полную производительность.

Синтаксис

```
trace-cmd reset [OPTIONS]
```

Описание

Команда `trace-cmd reset` выключает все трассировки `ftrace`, возвращая полную производительность системы, которая была до включения трассировки. Это нужно делать, поскольку команды `trace-cmd record`, `trace-cmd stop` и `trace-cmd extract` не отключают трассировщик даже после извлечения данных из буфера. Причина этого заключается в том, что пользователь может просто захотеть включить трассировщик вручную с использованием псевдофайловой системы или проверить другие части `ftrace` на предмет результатов работы `trace-cmd`. После команды `reset` все данные в кольцевых буферах и установленные опции будут потеряны.

Опции

Следует отметить, что для этой команды важен порядок указания опций в командной строке (см. Примеры).

-b buffer_size

При загрузке ядра кольцевой буфер `ftrace` имеет минимальный размер (3 страницы на CPU). При первом использовании трассировщика кольцевой буфер расширяется до установленного значения (по умолчанию 1,4 Мбайт на CPU).

Если дальнейшей трассировки не предполагается, эта опция позволяет уменьшить кольцевой буфер, сохраняя память.

```
trace-cmd reset -b 1
```

Задаёт воздействие на экземпляры буфера, заданные последней опцией `-B`, `-t` или `-a`:

при использовании после `-B`, изменяет размер экземпляра буфера, предшествующего опции в командной строке;

при использовании после `-a` меняет размер всех экземпляров буферов, за исключением верхнего уровня;

при использовании после `-t` или перед `-B` или `-a`, меняет размер буфера верхнего уровня.

-B buffer-name

Если ядро поддерживает несколько буферов, эта опция будет задавать воздействие лишь на указанный в ней буфер. Опция может включаться несколько раз для разных буферов. Буфер верхнего уровня не будет сбрасываться, если не указана также опция `-t`.

-a

Сбрасывает трассировку для всех имеющихся экземпляров буферов. При указании этой опции экземпляр верхнего уровня не сбрасывается, если не была задана также опция `-t`.

-d

Эта опция удаляет экземпляры буферов, заданные последними опциями `-B` или `-a`. Поскольку буфер верхнего уровня не может быть удалён, эту опцию нельзя указывать после опции `-t` или перед опцией `-B` или `-a` в командной строке.

-t

Сбрасывает экземпляр буфер верхнего уровня. Без опции `-B` или `-a` эта опция обеспечивает принятое по умолчанию поведение, но при наличии `-B` или `-a`, будет сбрасываться также буфер верхнего уровня.

Примеры

Для сброса трассировки в `instance-one` и установки размера буфера 4096 Кбайт на процессор, а также удаления `instance-two` служит приведённая ниже команда. Экземпляр верхнего уровня и другие буферы сохраняются.

```
trace-cmd reset -B instance-one -b 4096 -B instance-two -d
```

Для удаления всех экземпляров кроме буфера верхнего уровня служит приведённая ниже команда.

```
trace-cmd reset -a -d
```

Для удаления всех экземпляров и буфера верхнего уровня служит команда

```
trace-cmd reset -t -a -d
```

Приведённая ниже команда неверна, поскольку она пытается удалить экземпляр верхнего уровня.

```
trace-cmd reset -a -t -d
```

Для сброса экземпляра верхнего уровня и установки размера буфера 1024 Кбайт на процессор служит приведённая ниже команда. На другие буферы она не воздействует.

```
trace-cmd reset -b 1024
```

split

Расщепляет файл `trace.dat` на более мелкие файлы.

Синтаксис

```
trace-cmd split [OPTIONS] [start-time [end-time]]
```

Описание

Команда trace-cmd служит для разбиения trace.dat на более мелкие файлы. Параметр start-time указывает время, с которого начнётся новый файл. Команду trace-cmd report и копирование временной метки определённого события можно использовать в качестве start-time или end-time. Расщепление прекратит создание файлов по достижении времени end-time. Если нужно лишь время окончания, в качестве start-time служит 0.0.

Если параметр start-time опущен, расщепление начинается с начала файла. Если опущен параметр end-time, расщепление будет продолжаться до конца файла, если его не остановят другие опции.

Опции**-i file**

Если эта опция не задана, команда будет расщеплять файл trace.dat. С помощью опции можно указать иной файл.

-o file

По умолчанию команда split будет использовать имя входного файла в качестве основы для имён создаваемых файлов, добавляя к входному имени суффиксы вида .# (trace.dat.1, trace.dat.2 и т. п.). Данная опция позволяет изменить базовое имя файлов. Например, -o приведёт к созданию файлов file.1, file.2 и т. д.

-s seconds

Задаёт продолжительность времени (в секундах), по истечении которого запись в файл следует остановить и начать следующий.

-m milliseconds

Задаёт продолжительность времени (в миллисекундах), по истечении которого запись в файл следует остановить и начать следующий.

-u microseconds

Задаёт продолжительность времени (в микросекундах), по истечении которого запись в файл следует остановить и начать следующий.

-e events

Задаёт число событий, записываемых в файл прежде, чем начать новый.

-p pages

Задаёт число страниц, записываемых в файл прежде, чем начать новый.

Примечание. Опции -r, -e, -u, -m и -s являются взаимоисключающими, т. е. не может присутствовать более одной из них.

При установке опции -r автоматически добавляется опция -s.

-r

Эта опция заставляет повторять разбиение, пока не будет достигнуто значение end-time (или конец файла при отсутствии end-time).

```
trace-cmd split -r -e 10000
```

Будет разбивать trace.dat на файлы, содержащие не более 10000 событий в каждом.

-c

Эта опция приводит к разбиению файла по процессорам.

```
trace-cmd split -c -p 10
```

будет создавать файл, имеющий 10 страниц на каждый CPU.

-C cpi

Эта опция будет обеспечивать расщепление для одного CPU, извлекая из файла лишь данные, соответствующие параметру cpi.

```
trace-cmd split -C 1
```

будет выделять в файл все события для cpi 1.

list

Выводит список доступных плагинов, событий и опций ftrace.

Синтаксис

```
trace-cmd list [OPTIONS]
```

Описание

Команда trace-cmd list выводит список доступных плагинов, событий и опций ftrace, которые настроены на текущей машине. Если команда используется без опций, выводятся все плагины, события и опции ftrace на стандартный вывод.

Опции**-e [regex]**

Эта опция задаёт вывод всех доступных событий, разрешённых на локальной машине. Она может принимать в качестве дополнительного аргумента регулярное выражение, соответствующее regexp, для поиска событий.

```
trace-cmd list -e '^sys.*'
```

-F

Применяется вместе с -e regex для вывода форматов событий.

-I

Применяется вместе с -e regex для вывода фильтров событий.

-R

Применяется вместе с -e regex для вывода триггеров событий.

-t

Обеспечивает вывод списка доступных трассировщиков, разрешённых на локальной системе.

-p

Совпадает с -t и применяется для унаследованных систем.

-o

Обеспечивает вывод списка доступных опций `fttrace`, настроенных на локальной системе.

-f [regex]

Обеспечивает вывод всех доступных функций фильтрации. Это будет список функций системы, которые можно трассировать или фильтровать. Опция может принимать в качестве дополнительного аргумента регулярное выражение, соответствующее `regcomp`, для поиска функций.

```
trace-cmd list -f '^sched.*'
```

-P

Обеспечивает вывод списка файлов плагинов, загружаемых `trace-cmd report`.

-O

Обеспечивает вывод списка опций плагинов, которые могут применяться в команде `trace-cmd report -O option`.

-B

Выводит список экземпляров буферов.

-C

Выводит список часов, которые могут применяться в команде `trace-cmd record -C`. Активные часы указываются в квадратных скобках [].

listen

Задаёт прослушивание входящих соединений для записи трассировки.

Синтаксис

```
trace-cmd listen -p port [OPTIONS]
```

Описание

Команда `trace-cmd listen` активизирует порт для прослушивания входящих соединений от других хостов, на которых используется команда `trace-cmd record` с опцией `-N`. Когда соединение организовано, удалённый хост будет передавать через него данные, которые будут сохраняться в файле с именем `trace.HOST:PORT.dat` (`HOST` - имя удалённого хоста, `PORT` - номер порта, используемый удалённым хостом для соединения).

Опции

-p port

Эта опция задаёт порт для прослушивания входящих соединений.

-D

Эта опция переводит `trace-cmd listen` в режим демона.

-d dir

Эта опция задаёт каталог для записи файлов данных.

-o filename

Эта опция заменяет принятую по умолчанию запись в файл `trace.HOST:PORT.dat` записью в указанный файл.

-l filename

Эта опция задаёт запись выходных сообщений в указанный файл вместо стандартного вывода.

restore

Восстанавливает отказавшую запись трассировки.

Синтаксис

```
trace-cmd restore [OPTIONS] [command] cpu-file [cpu-file ...]
```

Описание

Команда `trace-cmd restore` восстанавливает файл после отказа `trace-cmd record`. Если при работе `trace-cmd record` возникает отказ, программа оставляет файлы данных для отдельных процессоров, не создавая финального файла `trace.dat`. Команда `trace-cmd restore` будет собирать эти файлы для создания рабочего файла `trace.dat`, который может быть прочитан командой `trace-cmd-report`.

При работе команды `trace-cmd record` она запускает отдельный процесс для каждого CPU и записывает для них файлы данных, обычно называемые `trace.dat.cpuX`, где `X` указывает номер процессора. Если для команды `trace-cmd record` использовалась опция `-o`, файлы данных CPU будут иметь другие имена вместо `trace.dat`. При возникновении отказа до завершения трассировки файлы отдельных CPU сохраняются, но файла `trace.dat` не будет. Команда `trace-cmd restore` создаёт `trace.dat` из имеющихся файлов данных.

Опции

-c

Задаёт создание частичного файла `trace.dat` для машины, который можно потом использовать в команде `trace-cmd restore`. Эта опция полезна для встраиваемых систем. Если сервер содержит файлы отдельных процессоров отказавшей команды `trace-cmd record` (или `trace-cmd listen`), `trace-cmd restore` можно запустить на встраиваемом устройстве с опцией `-c` для получения всех данных этого встраиваемого устройства. Затем созданный файл копируется на сервер для запуска команды `trace-cmd restore`.

Если опция `-o` не задана, создаётся файл `trace-partial.dat`. Это связано с тем, что файл не будет полной версией, пригодной для использования в `trace-cmd report`.

-t tracing_dir

При использовании вместе с опцией `-c` переопределяет место для считывания данных о событиях. По умолчанию данные трассировки считываются из каталога `debugfs/tracing`, но опция `-t` позволяет указать иное место. Это может быть полезно при создании файла `trace.dat` с другой машины.

-k kallsyms

При использовании вместе с опцией `-c` переопределяет место для считывания данных `kallsyms`. По умолчанию используется `/proc/kallsyms`, но опция `-k` позволяет указать иное место. Это может быть полезно при создании файла `trace.dat` с другой машины.

-o output

По умолчанию trace-cmd restore будет создавать файл trace.dat (или trace-partial.dat при наличии опции -c), но опция -o позволяет указать другой файл.

-i input

По умолчанию trace-cmd restore читает информацию текущей системы для создания начальных данных, хранящихся в файле trace.dat. Если отказ произошёл на другой машине, на ней следует запустить команду trace-cmd с опцией -c для создания частичного файла trace.dat. Затем этот файл копируется на машину, где используется команда trace-cmd restore с опцией -i для загрузки данных из файла вместо считывания из локальной системы.

Примеры

Если отказ произошёл на другой машине, можно воспользоваться командой

```
$ trace-cmd restore -c -o box-partial.dat
```

Затем на сервер, где размещены файлы данных сри используется команда

```
$ trace-cmd restore -i box-partial.dat trace.dat.cpu0 trace.dat.cpu1
```

Это позволяет восстановить файл trace.dat для встраиваемого устройства.

stack

Читает, включает или отключает трассировку ядра Linux ftrace.

Синтаксис

```
trace-cmd stack
```

Описание

Команда trace-cmd stack включает трассировщик стека в ядре, который включает трассировщик функций и при каждом вызове функции в ядре проверяется стек. При обнаружении нового максимума использования стека это записывается.

При отсутствии опции отображается текущий стек.

Для включения трассировщика стека используется опция --start, для отключения - --stop. Выводиться будет максимальный стек, найденный с момента старта.

Для сброса счётчика стека служит опция --reset.

check-events

Анализирует форматы событий в локальной системе.

Синтаксис

```
trace-cmd check-events [OPTIONS]
```

Описание

Команда trace-cmd check-events разбирает строки форматов для всех событий в локальной системе. Она возвращает информацию о возможности корректного анализа каждого формата. Если явно не указано иное, команда будет загружать плагины.

Эта команда полезна для проверки всех строк формата трассировки, которые могут включать внутренние функции ядра, не декодируемые вне ядра. Это может означать необходимость изменения необработанных строк формата или создания плагина для их анализа.

Опции

-N

Отменяет загрузку плагинов.

stream

Направляет трассировку в stdout.

Синтаксис

```
trace-cmd stream [OPTIONS] [command]
```

Описание

Команда trace-cmd запускает трассировку подобно trace-cmd record, но не выводит данные в файл, а считывает их напрямую из двоичного буфера, преобразуя в понятный человеку формат и выводит на stdout.

В основном это совпадает с командой trace-cmd start и последующей командой trace-cmd show с опцией -p. Команда trace-cmd stream не так эффективна как чтение из канала, поскольку большая часть потока выполняется в пользовательском пространстве. Эта команда полезна в тех случаях. Когда нужно выполнить основную работу в пользовательском пространстве, а не в ядре, а также помогает отлаживать команды trace-cmd profile, использующие код потока для анализа данных профиля в реальном масштабе времени.

Опции

Опции команды совпадают с опциями trace-cmd record за исключением -o.

snapshot

Принимает, сбрасывает, освобождает и выводит моментальные снимки трассировки ftrace (snapshot).

Синтаксис

Описание

Команда `trace-cmd snapshot` управляет или показывает моментальный снимок трассировки `ftrace` (если ядро поддерживает это). Она полезна для «замораживания» экземпляра трассировки без остановки процесса.

```
trace-cmd start -p function
trace-cmd snapshot -s
trace-cmd snapshot
[ выводит содержимое буфера в момент trace-cmd snapshot -s ]
trace-cmd snapshot -s
trace-cmd snapshot
[ выводит новое содержимое буфера при последней операции -s ]
```

Опции

-s

Делает моментальный снимок работающего буфера.

-r

Очищает буфер.

-f

Освобождает буфер моментальных снимков, использующий память ядра. Неиспользуемые буферы лучше освобождать. Первая опция `-s` будет выделять буфер, если он ещё не создан.

-c *cpu*

Задаёт вывод моментального снимка для отдельного процессора (может поддерживаться ядром не полностью)

-B *buf*

Если экземпляр буфера создан, опция `-B` будет работать со снимком в этом буфере.

Формат файла `trace-cmd.dat`

Описание

Утилита `trace-cmd` создаёт файл `trace.dat`. Этот файл может иметь другое имя, если пользователь указал его, но в любом случае он будет иметь некий двоичный формат. Файл применяется программой `trace-cmd` для хранения трассировок ядра с возможностью извлечения данных (см. `report`).

Начальный формат

Первые три байта файла содержат «магическое значение»

```
0x17 0x08 0x44
```

В следующий 7 байтах содержится строка

```
tracing
```

Далее следует строка символов с завершающим `\0`, указывающая версию файла. Например,

```
"6\0"
```

Следующий байт указывает тип значений

```
0 = little endian
1 = big endian
```

Далее следует байт, указывающий размер значения типа `long`

```
4 - 32-битовые значения long
8 - 64-битовые значения long
```

Примечание. Этот размер `long` предназначен для пользовательского пространства и не связан с размером в ядре.

Далее все числа указываются в зависимости от заданного порядка байтов (endianess).

Следующие 4 байта образуют 32-битовое слово, определяющее размер страницы на трассируемом хосте.

Формат данных заголовка

Сразу после начального формата следуют данные о заголовках трассировки, записанные от целевого устройства.

Следующие 12 байтов содержат строку

```
header_page\0
```

Затем следуют 8 байтов, образующих 64-битовое слово размера данных заголовка страницы, которая храниться далее.

Следующий блок данных имеет размер, указанный в предыдущих 8 байтах и содержит данные из `debugfs/tracing/events/header_page`.

Примечание. Размер второго поля `commit` содержит размер `long` в целевом ядре. Например,

```
field: local_t commit;      offset:8;      \fbsize:8;\fr signed:1;
```

указывает, что в ядре используются 64-битовые значения `long`.

Следующие 13 байтов содержат строку

```
header_event\0
```

Затем следуют 8 байтов, образующих 64-битовое слово размера заголовка данных событий, хранящегося далее.

Следующий блок, размер которого указан предыдущими 8 байтами, содержит данные из `debugfs/tracing/events/header_event`.

Эти данные позволяют `trace-cmd` узнать о наличии каких-либо изменений в кольцевом буфере ядра.

Формат событий ftrace

Сразу после заголовка следует информация о конкретных событиях трассировки, которые используются плагинами ftrace plugins и не включены трассировкой событий.

Первые 4 байта содержат 32-битовое слово числа файлов формата событий ftrace, сохранённых в файле.

Для числа вхождений, указанного предыдущими 4 байтами, 8 байтов указывают размер файла формата событий ftrace. Файл формата событий ftrace копируется с целевой машины

```
debugfs/tracing/events/ftrace/<event>/format
```

Формат событий

Сразу после форматов ftrace размещается информация о схеме событий.

Первые 4 байта содержат 32-битовое слово числа систем событий, хранящихся в файле. Это каталоги в структуре debugfs/tracing/events за исключением каталога ftrace.

Для числа вхождений, указанного предыдущими 4 байтами, указывается строка с null-символом в конце, задающая имя системы, 4 байта 32-битового слова числа событий в системе.

Для числа вхождений, указанного предыдущими 4 байтами, 8 байтов указывают размер файла формата событий.

Файл формата событий ftrace копируется с целевой машины

```
debugfs/tracing/events/<system>/<event>/format
```

Информация KALLSYMS

Сразу за форматами событий следуют данные отображения имён функций на их адреса.

Первые 4 байта содержат 32-битовое слово размера данных об отображении функций.

Следующий блок, размер которого определён предыдущими 4 байтами, содержит данные из файла целевой машины

```
/proc/kallsyms
```

Информация TRACE_PRINTK

Если разработчики применяли в ядре trace_printk(), можно сохранять строку формата вне кольцевого буфера, в файле

```
debugfs/tracing/printk_formats
```

Следующие 4 байта содержат 32-битовое слово размера данных, хранящихся в формате printk.

Далее следует блок, размер которого задан предыдущими 4 байтами, с данными из debugfs/tracing/printk_formats.

Данные процессов

Сразу после форматов trace_printk следуют данные отображения PID на имена процессов.

Следующие 8 содержат 64-битовое слово размера данных отображения PID на имена процессов.

Далее следует блок, размер которого задан предыдущими 8 байтами, с данными из debugfs/tracing/saved_cmdlines.

Остальная часть заголовка TRACE-CMD

Сразу после данных о процессах следует последний бит заголовка trace.dat.

Следующие 4 байта содержат 32-битовое слово числа CPU, обнаруженных на целевой машине (и имеющих данные трассировки).

Следующие 10 байтов содержат одну из строк

```
options \0
latency \0
flyrecord\0
```

Для случая options \0 следующие 2 байта содержат 16-битовое слово, задающее текущие опции (0 - больше нет опций).

В остальных случаях следующие 4 байта содержат 32-битовое слово размера опций. Если читающая сторона не понимает опцию, она может пропустить её. В настоящее время опции не определены, но здесь могут быть расширения данных.

Следующая опция размещается сразу за предыдущей и опции завершаются 0 в поле типа опции.

Следующие 10 байтов содержат одну из строк

```
latency \0
flyrecord\0
```

которые были бы и при отсутствии опций.

Для случая latency \0, остальная часть файла представляет просто текст ASCII из файла целевой системы

```
debugfs/tracing/trace
```

Для случая flyrecord\0 далее следует представленная ниже информация.

Для числа CPU, определённого раньше указывается:

8 байтов 64-битового слова, указывающего смещение в файле, где хранятся данные для CPU;

8 байтов 64-битового слова, указывающего размер данных CPU, расположенных по этому смещению.

Данные CPU

Данные процессоров хранятся в части файла, указанной в конце заголовка. Между заголовком и данными CPU помещается заполнение, для выравнивания данных CPU по краю страницы (страница цели) в файле.

Эти данные копируются напрямую из кольцевого буфера трассировки в том формате, который задан заголовком события, включённым в формат заголовка файла.

Программа trace-cmd будет пытаться выполнить mmap для данных страница за страницей с размером страниц целевой системы, если это возможно. При отказе mmap будут просто считываться данные.

По материалам руководств man

Николай Малых

nmalykh@protokols.ru