Трассировщик функций ftrace

Ftrace - это внутренний трассировщик ядра, позволяющий разработчикам посмотреть, что происходит внутри ядра системы. Трассировщик можно использовать для отладки, анализа задержек или или производительности, связанных с операциями за пределами пользовательского пространства.

Хотя ftrace обычно считают трассировщиком функций, реально это целый набор средств трассировки, включающий отслеживание задержки для просмотра событий между отключением и включением прерываний, при вытеснении задачи или с момента ее пробуждения до фактического планирования.

Одним из наиболее распространенных применений ftrace является отслеживание событий. В ядре имется сотни статических событий, которые могут быть включены через файловую систему tracefs для просмотра происходящего в отдельных частях ядра.

Дополнительную информацию о событиях можно найти в файле events.txt документации ядра.

Детали реализации трассировщика описаны в файле ftrace-design.

Оглавление

Файловая система

Ftrace использует файловую систему tracefs для хранения управляющих файлов и данных.

При настройке tracefs в ядре (выбор любой из опций ftrace) создается каталог /sys/kernel/tracing. Для автоматического монтирования этого каталога можно добавить в файл /etc/fstab команду

```
tracefs   /sys/kernel/tracing tracefs defaults    0         0
```

Можно монтировать файловую систему по мере надобности с помощью команды

```
mount -t tracefs nodev /sys/kernel/tracing
```

Для быстрого доступа к каталогу можно создать символьную ссылку

```
ln -s /sys/kernel/tracing /tracing
```

> Примечание. До версии ядра 4.1 управление трассировкой происходило в рамках файловой системы debugfs, обычно называемой /sys/kernel/debug/tracing. Для совместимости с прежними версиями при монтировании файловой системы debugfs автоматически монтируется система tracefs как /sys/kernel/debug/tracing[1].
>
> Все файлы системы tracefs доступны и в файловой системе debugfs.
>
> Все выбранные опции ftrace будут отражаться и в файловой системе tracefs. Далее в документе предполагается текущим каталог ftrace (cd /sys/kernel/tracing) и файлы будут именоваться относительно этого каталога без указания полного пути /sys/kernel/tracing.

Исходя из того, что настройка ftrace в ядре присутствует, после монтирования tracefs вы получите доступ к управлению и выходным файлам ftrace. Ниже описаны некоторые из этих файлов.

[1]Следует отметить, что в некоторых вариантах Linux каталог /sys/kernel/tracing остается пустым, а все упомянутые здесь файлы размещаются только в /sys/kernel/debug/tracing

current_tracer

Указывает текущий настроенный трассировщик.

available_tracers

Содержит список трассировщиков, включенных в ядре. Эти трассировщики можно настраивать, помещая имя нужного трассировщика с помощью команды echo в файл current_tracer.

tracing_on

Устанавливает или показывает состояние записи в кольцевой буфер трассировки. Запись 0 в этот файл отключает запись, 1 включает. Отметим, что это влияет лишь на запись в кольцевой буфер, а связанные с трассировкой издержки могут сохраняться.

Функция ядра tracing_off() может применяться внутри ядра для отключения записи в кольцевой буфер, помещая 0 в указанный файл. Можно восстановить запись из пользовательского пространства, отправив в файл значение 1.

Отметим, что функция и триггер событий traceoff также будут помещать в файл значение 0 и останавливать запись. Для восстановления записи служит указанный выше способ записи значения 1 в файл.

trace

В этот файл записывается вывод трассировки в понятном человеку формате (см. ниже). Отметим, что трассировка приостанавливается при чтении этого файла (open).

trace_pipe

Вывод в этот файл такой же как в trace, но файл предназначен для потоковой трассировки. Чтение из файла блокируется на время записи новых данных. В отличие от trace, этот файл является «потребителем» - считывание из этого файла «потребляет» данные и они уже не будут видны при последующих обращениях. Файл trace является «статическим» и если трассировщик не добавляет в него данных, при каждом обращении будет считываться одна и та же информация. При считывании файла trace_pipe трассировка не отключается.

trace_options

Этот файл позволяет пользователю контролировать объем данных, отображаемых в одном из указанных выше выходных файлов. Имеются также опции для изменения режима работы трассировщика и событий (трассировка стека, временные метки и . п.).

options

В этом каталоге расположены файлы для каждой доступной опции трассировки (те же, что в trace_options). Опции могут быть включены и отключены записью 1 или 0 (соответственно) в файл с именем опции.

tracing_max_latency

Некоторые из трассировщиков записывают максимальные задержки, например, максимальную продолжительность запрета прерываний. Данный файл служит для записи такой информации. Значение в файле меняется лишь в том случае, когда измеренная задержка превышает записанную ранее, т. е. значение в файле никогда не уменьшается, пока не будет сброшено. Задержка (и другие временные параметры, упоминаемые далее) указывается в микросекундах. Значение максимальной задержки сохраняется также в файле trace.

tracing_thresh

Некоторые трассировщики задержки записывают значение в файл, если оно превышает величину, указанную в этом файле (порог). Опция активна лишь при указании в файле значения больше 0 (в микросекундах).

buffer_size_kb

Размер буфера для каждого CPU в килобайтах. По умолчанию буферы трассировки имеют одинаковый размер для всех процессоров. Буферы трассировки выделяются страницами (блоками памяти, используемыми ядром; размер страницы обычно составляет 4 килобайта). Если последняя выделенная страница имеет размер больше запрошенного, реальный буфер будет использовать все пространство, а не то, которое запрошено и отображается.

Отметим, что размер буферов не обязан быть кратным размеру страницы из-за метаданных управления буферами.

Размеры буферов для отдельных CPU могут различаться (см. per_cpu/cpu0/buffer_size_kb ниже) и в этом случае файл будет содержать значение X.

buffer_total_size_kb

Общий размер буферов трассировки в килобайтах.

free_buffer

Если процесс выполняет трассировку и кольцевой буфер нужно «освободить» после завершения процесса, даже если процесс уничтожен по сигналу, лоя освобождения буфера можно использовать этот файл. При закрытии данного файла размер кольцевого буфера снижается до минимального. Выполняющий трассировку процесс открывает этот файл и при завершении процесса дескриптор файла будет освобождаться. В результате файл будет закрыт и кольцевой буфер «освободится».

При установке опции disable_on_free трассировка будет останавливаться.

tracing_cpumask

Маска, позволяющая пользователю задать трассировку лишь для нужных CPU. Маска указывается в шестнадцатеричном формате.

set_ftrace_filter

При динамической трассировке (см. dynamic ftrace ниже) код динамически изменяется для запрета вызова профилировщика функций (mcount). Это позволяет выполнить трассировку практически без влияния на производительность. Однако имеется побочное влияние на включение или отключение трассировки определенных функций. Отправка (echo) имен функций в этот файл ограничивает трассировку лишь включенными в список функциями. Это влияет на трассировщики function и function_graph, а также на профилировку функций (см. function_profile_enabled).

Функции, перечисленные в файле available_filter_functions могут быть записаны в файл фильтров.

Этот интерфейс можно использовать также для команд (см. Команды фильтрации).

set_ftrace_notrace

Это антипод set_ftrace_filter и любая функция, добавленная в файл, не будет трассироваться. При указании функции в set_ftrace_filter и set_ftrace_notrace трассировка этой функции выполняться **не будет**.

set_ftrace_pid

Указывает трассировщику function отслеживать лишь функции с PID, указанным в этом файле.

Если установлена опция function-fork, при ветвлении задачи с указанным в этом файле PID, значения PID дочерних процессов будут автоматически добавляться в этот файл и дочерние процессы также будут отслеживаться. При завершении задачи PID удаляется из файла.

set_event_pid

Задает трассировку событий лишь для процессов, PID указан в этом файле.

Отметим, что sched_switch и sched_wake_up также будут отслеживать события, указанные в этом файле.

Для добавления в файл PID дочерних процессо при ветвлении следует включить опцию event-fork (по завершении процесса PID будет удаляться из файла).

set_graph_function

Функции, указанные в этом файле и вызванные ими функции, будут отслеживаться трассировщиком function_ graph (см. Динамическая трассировка ftrace). Отметим, что set_ftrace_filter и set_ftrace_notrace сохраняют влияние на набор отслеживаемых функций.

set_graph_notrace

Похожа на set_graph_function, но отключает трассировщик function_graph до выхода из указанной в этом файле функции. Это позволяет пропустить трассировку функций, вызываемых из заданной функции.

available_filter_functions

Этот файл содержит список функций, для которых возможна трассировка, указанных по именам. Эти функции можно передать в опции (файлы) set_ftrace_filter, set_ftrace_notrace, set_graph_function и set_graph_notrace для управления их трассировкой (см. Динамическая трассировка ftrace)

dyn_ftrace_total_info

Этот файл служит для отладки и указывает число функций, которые были преобразованы в nop и доступны для трассировки.

enabled_functions

Этот файл больше подходит для отладки ftrace, но может быть полезен и для выяснения функций, с которыми связаны обратные вызовы (callback). Средства трассировки использует не только инфраструктура ftrace, но и другие подсистемы. В этом файле указаны все функции, с которыми связаны callback, а также число обратных вызовов. Отметим, что callback может также вызывать множество функций, которые не учитываются в этом числе.

Если обратный вызов зарегистрирован для трассировки функцией с атрибутом сохранения регистров (save regs), что ведет к добавочному росту издержек, в строке возвращающей регистры функции будет выводиться символ R.

Если обратный вызов зарегистрирован для трассировки функцией с атрибутом ip modify (т. е. regs->ip можно изменить), в строке с функцией, которая может быть переопределена, выводится символ I.

Если архитектура позволяет, будет также показано, для каких callback возможен прямой вызов из функции. Если значение счетчика больше 1, это скорей всего будет ftrace_ops_list_func().

Если обратный вызов функции имеет переход к trampoline[1], специфичному для callback, а не к стандартному trampoline, будет выводиться адрес перехода, а также функция, которую вызывает trampoline.

function_profile_enabled

При установке включает отслеживание всех функций с помощью трассировщика function или (если настроен) function_graph. Будет сохраняться гистограмма числа вызовов функций, а при настроенном трассировщике function_graph - еще и время, проведенное в этих функциях. Содержимое гистограммы размещается в файлах по процессорам trace_stat/function<cpu> (function0, function1 и т. д.).

trace_stat

Каталог, в котором хранится статистика трассировки.

kprobe_events

Включает точки динамической трассировки (см. файл kprobetrace.txt в документации ядра).

kprobe_profile

Статистика точек динамической трассировки (см. файл kprobetrace.txt в документации ядра).

max_graph_depth

Применяется с трассировщиком function_graph, задавая максимальную глубину трассировки внутрь функции. При установке в файле значения 1 будет показываться только первый вызов функции ядра из пользовательского пространства.

printk_formats

Этот файл служит для инструментов, читающих необработанные (raw) файлы. Если событие в кольцевом буфере указывается строкой, в буфер записывается лишь указатель, а не сама строка. Это не позволяет инструментам узнать, что было в строке. Данный файл отображает строки и адреса, позволяя инструментам сопоставлять указатели со строками.

saved_cmdlines

В трек событий записывается лишь pid задачи, если событие специально не сохраняет и команду. Ftrace кэширует отображения pid на команды, чтобы попытаться указать для событий и команду. Если pid для команды не указа, выводится <...>.

Если для опции record-cmd установлено значение 0, команды задач не сохраняются при записи. По умолчанию сохранение включено.

saved_cmdlines_size

По умолчанию сохраняется 128 команд (см. saved_cmdlines выше). Для изменения числа кэшируемых команд в этот файл записывается соответствующее число.

saved_tgids

При установленной опции record-tgid для каждого запланированного переключения контекста сохраняется идентификатор группы (Task Group ID) для задачи в таблице сопоставления PID потока с TGID. По умолчанию опция record-tgid выключена.

snapshot

Показывает «моментальный снимок» (snapshot) буфера и позволяет пользователю сделать снимок текущей трассировки (см. раздел Мгновенные снимки).

stack_max_size

При активизации трассировщика стека здесь будет указан максимальный наблюдаемый размер стека (см. раздел Трассировка стека).

[1]В GCC термин trampoline означает метод реализации указателей на вложенные функции. Trampoline - это небольшой фрагмент кода, который создается «на лету» в стеке, когда береься адрес вложенной функции. Trampoline устанавливает статический указатель на привязку (link), который позволяет вложенной функции обращаться к переменным вмещающей ее функции. Указателем на функцию является просто адрес trampoline. Это избавляет от применения «толстых» указателей на функции, передающих адрес кода и статическую ссылку. Однако это вступает в противоречие с тенденцией отказат от исполняемого стека по соображениям безопасности.

stack_trace

Здесь указывается обратная трассировка наибольшего стека, который встретился при активизации трассировщика стека (см. раздел Трассировка стека).

stack_trace_filter

Похоже на set_ftrace_filter, но ограничивает функции, которые будет проверять трассировщик стека.

trace_clock

При записи события в кольцевой буфер указывается временная метка, которая берется из определенных часов. По умолчанию ftrace использует локальные часы (local). Это очень быстрые часы, привязанные к каждому процессору, но они могут быть не синхронизированы между разными CPU.

```
# cat trace_clock
[local] global counter x86-tsc
```

Используемые часы указываются в квадратных скобках.

local

Используемые по умолчанию часы, которые могут быть не синхронизированы между CPU.

global

Эти часы синхронизированы на всех CPU, но могут быть медленней часов local.

counter

Это не часы, а счетчик, увеличивающий значение на 1 и синхронизированный со всеми CPU. Счетчик может быть полезен, когда нужно знать точный порядок событий, связанных с разными CPU.

uptime

Это счетчик циклов jiffy и временная метка относительно момента последней загрузки системы.

perf

Задает для ftrace использование тех же часов, которые применяются perf. В конечном счете perf сможет считывать буферы ftrace buffers и это поможет при чередовании данных.

x86-tsc

Архитектура может задавать свои часы. Например, в x86 применяются часы TSC.

ppc-tb

Значение регистра timebase в powerpc. Оно синхронизировано для всех CPU и может служить для сопоставления событий между гипервизором и гостевыми системами, если известно значение tb_offset.

mono

Быстрый, монотонно возрастающий счетчик (CLOCK_MONOTONIC), согласуемый с NTP .

mono_raw

Быстрый, монотонно возрастающий счетчик (CLOCK_MONOTONIC_RAW) без возможности согласования частоты, работающий от аппаратного генератора сигналов.

boot

Загрузочные часы (CLOCK_BOOTTIME) на основе быстрого монотонно возрастающего счетчика, учитывающие время ожидания. Поскольку доступ к этим часам рассчитан на использование при трассировке на пути с остановками, возможны побочные эффекты при доступе к часам после того, как было учтено время приостановки до обновления монотонно возрастающего счетчика. В 32-битовых системах 64-битовое смещение загрузки может обновляться частично. Эти эффекты возникают редко и постобработка может справляться с ними (см. комментарии в функции ядра ktime_get_boot_fast_ns()).

Для выбора часов просто помещается их имя в файл trace_clock

```
# echo global > trace_clock
```

trace_marker

Этот файл очень полезен для синхронизации пользовательского пространства с событиями в ядре. Запись строки в этот файл копируется в буфер трассировки ftrace.

Полезно открыть этот файл в приложении при его запуске и просто сослаться на дескриптор файла

```
void trace_write(const char *fmt, ...)
{
        va_list ap;
        char buf[256];
        int n;

        if (trace_fd < 0)
                return;

        va_start(ap, fmt);
        n = vsnprintf(buf, 256, fmt, ap);
        va_end(ap);

        write(trace_fd, buf, n);
}

start::

        trace_fd = open("trace_marker", WR_ONLY);
```

Примечание. Запись в файл trace_marker будет также вызывать запись в /sys/kernel/tracing/events/ftrace/print/trigger (см. Event triggers в файле Documentation/trace/events.rst и пример в разделе 3 файла trace/histogram.rst)

trace_marker_raw

Похож на trace_marker, но предназначен для записи двоичных данных, когда можно использовать инструмент для анализа.

uprobe_events

Добавляет в программу динамические точки трассировки (см. uprobetracer.txt).

uprobe_profile

Статистика uprobe (см. uprobetracer.txt).

instances

Это способ создания множества буферов трассировки, куда могут записываться разные события (см. Экземпляры).

events

Это каталог событий трассировки, в котором хранятся события точек трассировки (статические точки трассировки), встроенных в ядро. Это показывает наличие точек трассировки и их группировку в системе. Здесь расположены файлы «разрешения» для различных уровней, которые позволяют включить точки трассировки при записи 1 в соответствующий файл (см. events.txt).

set_event

Запись события в этот файл будет разрешать трассировку этого события (см. events.txt).

available_events

Список событий, трассировка которых может быть включена (см. events.txt).

timestamp_mode

Некоторые трассировщики могут менять режим временных меток, используемый при записи событий в кольцевой буфер. События с разными режимами меток могут сосуществовать в буфере, но действующий режим влияет на временную метку, записываемую с событием. По умолчанию применяется режим меток delta (приращение).

```
# cat timestamp_mode
[delta] absolute
```

В квадратных скобках указывается действующий режим временных меток.

delta

Указывается интервал времени от предшествующего события в данном буфере. Применяется по умолчанию.

absolute

Временная метка содержит полное время, а не приращение от предыдущего значения. Эти метки занимают больше места и менее эффективны.

hwlat_detector

Каталог для детектора задержек в оборудовании (см. Определение аппаратной задержки).

per_cpu

Каталог с данными трассировки для каждого процессора. В качестве примера показан процессор 0.

per_cpu/cpu0/buffer_size_kb

Размер буфера ftrace для данного процессора. Отдельные буферы позволяют каждому процессору записывать данные самостоятельно без общего кэширования. Буферы разных процессоров могут различаться по размеру. Этот файл поход на buffer_size_kb file, но показывает размер буфера отдельного процессора.

per_cpu/cpu0/trace

Похож на файл trace, но отображает данные лишь для одного CPU. При записи очищается буфер только данного CPU.

per_cpu/cpu0/trace_pipe

Похож на trace_pipe и «потребляет» данные при чтении, но отображает (и потребляет) данные лишь одного CPU.

per_cpu/cpu0/trace_pipe_raw

Для инструментов, понимающих двоичный формат кольцевого буфера ftrace, файл trace_pipe_raw может служить для извлечения информации напрямую из кольцевого буфера. С помощью системного вызова splice() данные из буфера можно быстро перенести в файл или сеть для сбора на сервере.

Подобно trace_pipe, это потребляющий считыватель, где каждое чтение будет возвращать новые данные.

per_cpu/cpu0/snapshot

Похож на файл snapshot, но дает снимок только для одного CPU (если это поддерживается). При записи в файл очищается только буфер данного CPU.

per_cpu/cpu0/snapshot_raw

Похож на trace_pipe_raw, но считывает двоичные данные лишь из «моментального снимка» для данного CPU.

per_cpu/cpu0/stats

Статистика из кольцевого буфера процессора.

entries

Число событий, остающихся в буфере.

overrun

Число перезаписанных событий в результате заполнения буфера.

commit overrun

Значение в файле устанавливается, если произошло много событий внутри вложенного события (повторный вход в кольцевой буфер), заполнивших буфер и приведших к отбрасыванию событий. В нормальных условиях содержит 0.

bytes

Число реально прочитанных байтов (не перезаписанных).

oldest event ts

Временная метка самого старого события в буфере.

now ts

Текущая временная метка.

dropped events

События, потерянные в результате выключения опции перезаписи.

read events

Число считываний событий.

Трассировщики

Ниже приведен список трассировщиков, которые могут быть настроены в системе.

function

Трассировщик вызовов для всех функций ядра.

function_graph

Похож на трассировщик function, но отличается дополнительным отслеживанием выхода из функции, которое первый не выполняет. Это позволяет построить граф вызовов функций, похожий на исходный код языка C.

blk

Трассировщик блоков, применяемый пользовательским приложением blktrace.

hwlat

Трассировщик аппаратной задержки, служащий для обнаружения задержки в оборудовании. (см. Определение аппаратной задержки).

irqsoff

Трассирует области, где запрещены прерывания и сохранеяет вызов с самой большой максимальной задержкой, заменяя имеющееся значение, если новый максимум превышает его. Лучше всего использовать эту трассировку со включенной опцией latency-format, которая автоматически включается при выборе трассировщика.

Этот трассировщик похож на preemptirqsoff, но отслеживает лишь максимальный интервал запрета прерываний.

preemptoff

Похож на трассировщик preemptirqsoff, но отслеживает лишь максимальный интервал запрета вытеснения.

preemptirqsoff

Похож на irqsoff и preemptoff, но трассирует и записывает лишь наибольшее время, когда irq и/или вытеснение были отключены.

wakeup

Трассировка максимальной задержки между пробуждением процесса с наивысшим приоритетом и запланированным его использованием.

wakeup_rt

Трассировка максимальной задержки между пробуждением процесса с наивысшим приоритетом и запланированным его использованием. Этот трассировщик отслеживает лишь задачи, работающие в режиме реального времени (RT[1]), а задачи SCHED_OTHER игнорируются.

wakeup_dl

Отслеживает и записывает максимальную задержку для задач SCHED_DEADLINE (подобно wakeup и wakeup_rt).

mmiotrace

Специальный трассировщий, служащий для отслеживания двоичных модулей. Он будет отслеживать все вызовы, которые модуль делает для оборудования.

branch

Этот трассировщик можно настроит на вероятные и маловероятные вызовы внутри ядра. Он будет отслеживать вероятные и маловероятные ветвления и оценивать корректность их предсказания.

nop

Отключает трассировку. Для выключения всех трассировщиков достаточно просто записать nop в файл current_tracer.

Примеры использования трассировщиков

Здесь представлены типовые примеры использования трассировщиков с управлением лишь через интерфейс tracefs (без утилит пользовательского пространства).

Выходной формат

Ниже представлен пример выходного формата файла трассировки (trace).

```
# tracer: function
#
# entries-in-buffer/entries-written: 140080/250280   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
            bash-1977  [000] .... 17284.993652: sys_close <-system_call_fastpath
            bash-1977  [000] .... 17284.993653: __close_fd <-sys_close
            bash-1977  [000] .... 17284.993653: _raw_spin_lock <-__close_fd
            sshd-1974  [003] .... 17284.993653: __srcu_read_unlock <-fsnotify
            bash-1977  [000] .... 17284.993654: add_preempt_count <-_raw_spin_lock
            bash-1977  [000] ...1 17284.993655: _raw_spin_unlock <-__close_fd
            bash-1977  [000] ...1 17284.993656: sub_preempt_count <-_raw_spin_unlock
            bash-1977  [000] .... 17284.993657: filp_close <-__close_fd
            bash-1977  [000] .... 17284.993657: dnotify_flush <-filp_close
            sshd-1974  [003] .... 17284.993658: sys_select <-system_call_fastpath
            ....
```

Заголовок выводится с именем трассировщика (в примере function). Затем выводится число событий в буфере и общее число сделанных записей. Разница между этими значениями показывает число потерянных событий (250280 - 140080 = 110200 потерянных событий).

В заголовке разъясняется содержимое событий. Имя задачи bash, ее идентификатор PID - 1977, задача работает на CPU 000, задержка (формат описан ниже, временная метка в формате <сек>.<мксек>, имя отслеживаемой функции (sys_close и родительская функция system_call_fastpath). Временная метка показывает момент входа в функцию.

Формат трассировки задержек

При включенной опции latency-format или установке одного из трассировщиков задержки файл trace содержит дополнительную информацию, позволяющую увидеть причину задержки. Ниже приведен пример трассировки.

```
# tracer: irqsoff
#
# irqsoff latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 259 us, #4/4, CPU#2 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: ps-6143 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: __lock_task_sighand
#  => ended at:   _raw_spin_unlock_irqrestore
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
      ps-6143    2d...    0us!: trace_hardirqs_off <-__lock_task_sighand
      ps-6143    2d..1  259us+: trace_hardirqs_on <-_raw_spin_unlock_irqrestore
      ps-6143    2d..1  263us+: time_hardirqs_on <-_raw_spin_unlock_irqrestore
      ps-6143    2d..1  306us : <stack trace>
 => trace_hardirqs_on_caller
 => trace_hardirqs_on
```

[1]Real time.

```
 => _raw_spin_unlock_irqrestore
 => do_task_stat
 => proc_tgid_stat
 => proc_single_show
 => seq_read
 => vfs_read
 => sys_read
 => system_call_fastpath
```

Вывод показывает, что трассировщик irqsoff отслеживает время, на которое отключались прерывания. Указана версия трассировки (никогда не меняется) и версия ядра (3.8). Затем указана максимальная задержка в микросекундах (259), число показанных событий и общее их число (#4/4). VP, KP, SP и HP всегда имеют значение 0 (резерв на будущее). #P указывает число работающих процессоров CPU (#P:4).

В качестве задачи указывается процесс, который работал при возникновении задержки (ps pid: 6143).

Строки started at и ended at указывают функции, в которых прерывания были отключены и включены (соответственно), что и привело к задержке.

- В функции __lock_task_sighand прерывания были отключены.
- В функции _raw_spin_unlock_irqrestore прерывания были снова включены.

Далее следует заголовок, разъясняющий формат вывода, и результаты трассировки.

cmd
Имя процесса в трассировке.
pid
PID этого процесса.
CPU#
Номер CPU, на котором выполнялся процесс.
irqs-off
Символ d указывает запрет прерываний, в остальных случаях выводится точка (.). Если архитектура не поддерживает считывание переменной флагов irq, в этом поле всегда указывается символ X.
need-resched
- N - установлены оба флага TIF_NEED_RESCHED и PREEMPT_NEED_RESCHED;
- n - установлен только флаг TIF_NEED_RESCHED;
- p - установлен только флаг PREEMPT_NEED_RESCHED;
- . - иное.

hardirq/softirq
- Z - прерывание NMI внутри hardirq;
- z - NMI выполняется;
- H - аппаратное irq внутри softirq;
- h - аппаратное irq выполняется;
- s - программное irq выполняется;
- . - обычный контекст.

preempt-depth
Уровень preempt_disabled.
Описанное выше предназначено в основном для разработчиков ядра.

time:
При включенной опции latency-format вывод файла trace включает временные метки относительно начала трассировки, а при выключенной опции latency-format метки указываются в абсолютном времени.
delay
Это просто метки для упрощения зрительного восприятия, относящиеся к одному CPU и показывающие разницу между текущей и следующей трассировкой.
- $ - больше 1 секунды;
- @ - больше 100 миллисекунд;
- * - больше 10 миллисекунд;
- # - больше 1000 микросекунд;
- ! - больше 100 микросекунд;
- + - больше 10 микросекунд;
- ' ' - не больше 10 микросекунд.

Остальная часть вывода такая же как для файла trace.

Отметим, что трассировщики задержки обычно завершаются трассировкой назад, чтобы было увидеть место задержки.

Опции трассировки - файл trace_options

Файл trace_options (или каталог options) служит для управления выводом трассировки или манипуляций с трассировщиками. Для просмотра доступного просто введите команду cat.

```
cat trace_options
        print-parent
        nosym-offset
        nosym-addr
        noverbose
        noraw
        nohex
        nobin
        noblock
        trace_printk
        annotate
        nouserstacktrace
        nosym-userobj
        noprintk-msg-only
        context-info
        nolatency-format
        record-cmd
        norecord-tgid
        overwrite
```

```
        nodisable_on_free
        irq-info
        markers
        noevent-fork
        function-trace
        nofunction-fork
        nodisplay-graph
        nostacktrace
        nobranch
```

Для отключения нужной опции добавьте к ее имени префикс no и отправьте в файл trace_options[1]

```
echo noprint-parent > trace_options
```

Для включения опции отправьте ее имя в файл trace_options

```
echo sym-offset > trace_options
```

Список доступных опций приведен ниже.

print-parent

При трассировке функций задает вывод вызывающей (родительской) функции вместе с трассируемой.

print-parent

```
bash-4000  [01]  1477.606694: simple_strtoul <-kstrtoul
```

noprint-parent

```
bash-4000  [01]  1477.606694: simple_strtoul
```

sym-offset

Вывод не только имени функции, но и ее смещения. Например, вместо ktime_get будет ktime_get+0xb/0x20.

sym-offset

```
bash-4000  [01]  1477.606694: simple_strtoul+0x6/0xa0
```

sym-addr

Вывод адреса функции вместе с ее именем.

sym-addr

```
bash-4000  [01]  1477.606694: simple_strtoul <c0339346>
```

verbose

Задает подробный вывод в файле trace при включенной опции latency-format.

```
 bash  4000 1 0 00000000 00010a95 [58127d26] 1720.415ms  (+0.000ms): simple_strtoul (kstrtoul)
```

raw

Вывод необработанных (raw) чисел. Эта опция удобна для использования с приложениями, транслирующими необработанные числа лучше, чем это делает ядро.

hex

Похожа на raw, но числа выводятся в шестнадцатеричном формате.

bin

Вывод в необработанном двоичном формате.

block

При установленной опции считывание файла trace_pipe не будет блокироваться при опросе.

trace_printk

Может запрещать запись trace_printk() в буфер.

annotate

При заполненных буферах CPU может возникать путаница, когда в одном буфере имеется много свежих событий, а у другого событий немного и буфер будет содержать более старые события. При выводе трассировки сначала указываются более старые события и может показаться, что работает лишь один процессор. При включенном аннотировании будет указываться запуск нового буфера CPU.

```
                          <idle>-0     [001] dNs4 21169.031481: wake_up_idle_cpu <-add_timer_on
                          <idle>-0     [001] dNs4 21169.031482: _raw_spin_unlock_irqrestore <-add_timer_on
                          <idle>-0     [001] .Ns4 21169.031484: sub_preempt_count <-_raw_spin_unlock_irqrestore
                ##### CPU 2 buffer started ####
                          <idle>-0     [002] .N.1 21169.031484: rcu_idle_exit <-cpu_idle
                          <idle>-0     [001] .Ns3 21169.031484: _raw_spin_unlock <-clocksource_watchdog
                          <idle>-0     [001] .Ns3 21169.031485: sub_preempt_count <-_raw_spin_unlock
```

userstacktrace

Эта опция меняет трассировку и ведет к записи трассировки стека (stacktrace) текущего потока пользовательского пространства после каждого трассируемого события.

sym-userobj

При включенной опции userstacktrace определяется, к какому объекту относится адрес и выводится относительный адрес. Это особенно полезно при включенном ASLR, поскольку иначе не будет возможности преобразовать адрес в объект/файл/строку после того, как приложение прекратит работу. Поиск выполняется при чтении trace или trace_pipe.

```
a.out-1623  [000] 40874.465068: /root/a.out[+0x480] <-/root/a.out[+0
x494] <- /root/a.out[+0x4a8] <- /lib/libc-2.7.so[+0x1e1a6]
```

printk-msg-only

При установке этой опции trace_printk() будет показывать только формат, а не параметры (если использовался вызов trace_bprintk() или trace_bputs() для сохранения trace_printk()).

context-info

Задает вывод только событий без команд, PID, временных меток, CPU и других данных.

latency-format

Эта опция меняет вывод трассировки и при включении обеспечивает дополнительную информацию о задержке, как описано в разделе Формат трассировки задержек.

record-cmd

При включении любого события или трассировщика в точке трассировки sched_switch включается ловушка для заполнения кэша сопоставлений pid и команд. Однако это может приводить к издержкам и если вам достаточно pid без имен задач, отключение этой опции может снизить влияние на трассировку (см. saved_cmdlines).

record-tgid

При включении любого события или трассировщика в точке трассировки sched_switch включается ловушка для заполнения кэша сопоставлений pid и TGID[2] (см. saved_tgids).

[1]Следует отметить, что, не смотря на одиночный символ >, содержимое файла не переписывается и меняется лишь строка указанной опции.

[2]Thread Group ID - идентификатор группы потоков.

overwrite

Управляет поведением при заполнении буфера трассировки. Значение 1 (принято по умолчанию) ведет к перезаписи самых старых событий, 0 - к перезаписи самых новых (см. per_cpu/cpu0/stats, где указаны переполнения и отбрасывания).

disable_on_free

При закрытии free_buffer трассировка будет прекращаться (устанавливается tracing_on = 0).

irq-info

Показывает прерывание, счетчик вытеснения и необходимость перепланировки данных. При отключенной опции трассировка будет иметь вид

```
# tracer: function
#
# entries-in-buffer/entries-written: 144405/9452052   #P:4
#
#           TASK-PID   CPU#      TIMESTAMP  FUNCTION
#              | |       |          |         |
           <idle>-0     [002]  23636.756054: ttwu_do_activate.constprop.89 <-try_to_wake_up
           <idle>-0     [002]  23636.756054: activate_task <-ttwu_do_activate.constprop.89
           <idle>-0     [002]  23636.756055: enqueue_task <-activate_task
```

markers

Установка опции открывает файл trace_marker для записи (пользователю root). При отключенной опции запись в trace_marker будет приводить к ошибке EINVAL.

event-fork

При установке опции для задач, PID которых указаны в set_event_pid, в этот файл будут добавляться при ветвлении PID дочерних процессов. При завершении задачи с PID, указанным в set_event_pid, значение PID будет удаляться из файла.

function-trace

При включенной опции трассировщики задержек будут включать трассировку функций (принято по умолчанию). Если опция не установлена при трассировке задержки функции отслеживаться не будут для снижения издержек.

function-fork

При установке опции для задач, PID которых указаны в set_ftrace_pid, в этот файл будут добавляться при ветвлении PID дочерних процессов. При завершении задачи с PID, указанным в set_ftrace_pid exit, значение PID будет удаляться из файла.

display-graph

При включенной опции трассировщик задержек (irqsoff, wakeup и т. п.) будет применять трассировку function_graph вместо function.

stacktrace

При включенной опции записывается трассировка стека после записи любого события.

branch

Разрешает трассировку ветвления, включая трассировщик branch вместе с текущим трассировщиком. Включение этой опции с трассировщиком nop эквивалентно простому включению трассировщика branch.

Примечание. У некоторых трассировщиков имеются свои опции, которые выводятся в файле только при их активизации. Однако они всегда присутствуют в каталоге options.

Ниже рассматриваются опции отдельных трассировщиков.

Опции трассировщика function

func_stack_trace

При установке этой опции записывается трассировка стека после записи каждой функции. **Перед включением этой опции важно ограничить число отслеживаемых функций с помощью фильтра set_ftrace_filter, поскольку иначе производительность системы сильно снизится.** Не забывает отключить эту опцию до отмены фильтра отслеживаемых функций.

Опции трассировщика function_graph

Поскольку трассировщик function_graph отличается выводом, у него имеются свои опции для контроля вывода.

funcgraph-overrun

При установке опции «переполнение» стека function_graph отображается после каждой отслеживаемой функции. Переполнение возникает, когда грубина стека вызовов превышает значение, зарезервированное для каждой задачи. Каждая задача имеет фиксированный массив функций для отслеживания в графе вызовов. Если глубина вызовов превышает предел, функция не отслеживается. Переполнение - это число функций, пропущенных в результате превышения массива.

funcgraph-cpu

При установке опции отображается номер CPU, где выполнялась трассировка.

funcgraph-overhead

При установленной опции отображается маркер задержки, если функция заняла время, превышающее некое значение.

funcgraph-proc

В отличие от других трассировщиков командная строка здесь по умолчанию не отображается и указывается лишь при отслеживании задачи во время переключения контекста. Включение этой опции ведет к выводу команды каждого процесса в каждой строке.

funcgraph-duration

В конце каждой функции (возврат) выводится продолжительность ее работы в микросекундах.

funcgraph-abstime

При установке опции в каждой строке выводится временная метка.

funcgraph-irqs

При отключенной опции функции внутри прерываний не отслеживаются.

funcgraph-tail

При установленной опции событие возврата будет включать представляемую им функцию. По умолчанию это отключено и при возврате из функции отображается лишь фигурная скобка }.

sleep-time

Служит для учета запланированного времени задачи при работе трассировщика function_graph. При включенной опции запланированное время будет учитываться как часть вызова функции.

graph-time

При запуске профилировщика функций с трассировщиком function_graph служит для включения времени вызова вложенных функций. Если опция не задана, указываемое для функции время будет включать только время выполнения самой функции без учета вызываемых ею функций.

Опции трассировщика blk

blk_classic

Задает минимальный вывод.

Трассировщик irqsoff

При отключенных прерываниях CPU не может реагировать на другие внешние события, кроме NMI и SMI. В результате замедляется реакция системы на события.

Трассировщик irqsoff отслеживает периоды запрета прерываний. При достижении нового значения максимальной задержки трассировщик сохраняет трассировку к точке задержки, чтобы при достижении нового максимума прежнее значения отбрасывалось, а новое записывалось.

Для сброса максимума записывается значение 0 в файл tracing_max_latency. Ниже приведен пример.

```
# echo 0 > options/function-trace
# echo irqsoff > current_tracer
# echo 1 > tracing_on
# echo 0 > tracing_max_latency
# ls -ltr
[...]
# echo 0 > tracing_on
# cat trace
# tracer: irqsoff
#
# irqsoff latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 16 us, #4/4, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: swapper/0-0 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: run_timer_softirq
#  => ended at:   run_timer_softirq
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
  <idle>-0       0d.s2    0us+: _raw_spin_lock_irq <-run_timer_softirq
  <idle>-0       0dNs3   17us : _raw_spin_unlock_irq <-run_timer_softirq
  <idle>-0       0dNs3   17us+: trace_hardirqs_on <-run_timer_softirq
  <idle>-0       0dNs3   25us : <stack trace>
 => _raw_spin_unlock_irq
 => run_timer_softirq
 => __do_softirq
 => call_softirq
 => do_softirq
 => irq_exit
 => smp_apic_timer_interrupt
 => apic_timer_interrupt
 => rcu_idle_exit
 => cpu_idle
 => rest_init
 => start_kernel
 => x86_64_start_reservations
 => x86_64_start_kernel
```

Вывод показывает задержку 16 мксек (очень хорошо). Прерывания отключены функцией _raw_spin_lock_irq в run_timer_softirq. Разница между задержкой в 16 мксек и показанной временной меткой 25us обусловлена инкрементированием часов между моментами записи максимальной задержки и функции, вызвавшей эту задержку.

Отметим, что в приведенном выше примере опция function-trace не была установлена и при ее установке вывод будет более подробным, как показано ниже.

```
with echo 1 > options/function-trace
 # tracer: irqsoff
 #
 # irqsoff latency trace v1.1.5 on 3.8.0-test+
 # --------------------------------------------------------------------
 # latency: 71 us, #168/168, CPU#3 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
 #    -----------------
 #    | task: bash-2042 (uid:0 nice:0 policy:0 rt_prio:0)
 #    -----------------
 #  => started at: ata_scsi_queuecmd
 #  => ended at:   ata_scsi_queuecmd
 #
 #
 #                  _------=> CPU#
 #                 / _-----=> irqs-off
 #                | / _----=> need-resched
 #                || / _---=> hardirq/softirq
 #                ||| / _--=> preempt-depth
 #                |||| /     delay
 #  cmd     pid   ||||| time  |   caller
 #     \   /      |||||  \    |   /
     bash-2042    3d...    0us : _raw_spin_lock_irqsave <-ata_scsi_queuecmd
     bash-2042    3d...    0us : add_preempt_count <-_raw_spin_lock_irqsave
     bash-2042    3d..1    1us : ata_scsi_find_dev <-ata_scsi_queuecmd
     bash-2042    3d..1    1us : __ata_scsi_find_dev <-ata_scsi_find_dev
     bash-2042    3d..1    2us : ata_find_dev.part.14 <-__ata_scsi_find_dev
     bash-2042    3d..1    2us : ata_qc_new_init <-__ata_scsi_queuecmd
     bash-2042    3d..1    3us : ata_sg_init <-__ata_scsi_queuecmd
     bash-2042    3d..1    4us : ata_scsi_rw_xlat <-__ata_scsi_queuecmd
```

```
    bash-2042    3d..1    4us : ata_build_rw_tf <-ata_scsi_rw_xlat
[...]
    bash-2042    3d..1   67us : delay_tsc <-__delay
    bash-2042    3d..1   67us : add_preempt_count <-delay_tsc
    bash-2042    3d..2   67us : sub_preempt_count <-delay_tsc
    bash-2042    3d..1   67us : add_preempt_count <-delay_tsc
    bash-2042    3d..2   68us : sub_preempt_count <-delay_tsc
    bash-2042    3d..1   68us+: ata_bmdma_start <-ata_bmdma_qc_issue
    bash-2042    3d..1   71us : _raw_spin_unlock_irqrestore <-ata_scsi_queuecmd
    bash-2042    3d..1   71us : _raw_spin_unlock_irqrestore <-ata_scsi_queuecmd
    bash-2042    3d..1   72us+: trace_hardirqs_on <-ata_scsi_queuecmd
    bash-2042    3d..1  120us : <stack trace>
 => _raw_spin_unlock_irqrestore
 => ata_scsi_queuecmd
 => scsi_dispatch_cmd
 => scsi_request_fn
 => __blk_run_queue_uncond
 => __blk_run_queue
 => blk_queue_bio
 => generic_make_request
 => submit_bio
 => submit_bh
 => __ext3_get_inode_loc
 => ext3_iget
 => ext3_lookup
 => lookup_real
 => __lookup_hash
 => walk_component
 => lookup_last
 => path_lookupat
 => filename_lookup
 => user_path_at_empty
 => user_path_at
 => vfs_fstatat
 => vfs_stat
 => sys_newstat
 => system_call_fastpath
```

Здесь показана задержка в 71 мксек, но видны функции, которые вызвались в это время. Отметим, что включение трассировки функций увеличило издержки, которые могли привести к росту задержки. Тем не менее, эта трассировка содержит много полезной для отладки информации.

Трассировщик preemptoff

При отключенном вытеснении (preemption) прерывания будут работать, но задачи не могут вытеснять друг друга и в результате задаче с высоким приоритетом может потребоваться ожидать возможности вытеснения менее приоритетной задачи.

Трассировщик preemptoff отслеживает места, где вытеснение запрещено. Подобно трассировщику irqsoff, он записывает максимальную задержку, для которой вытеснение было отключено. Управление трассировщиком preemptoff похоже на управление irqsoff.

```
# echo 0 > options/function-trace
# echo preemptoff > current_tracer
# echo 1 > tracing_on
# echo 0 > tracing_max_latency
# ls -ltr
[...]
# echo 0 > tracing_on
# cat trace
# tracer: preemptoff
#
# preemptoff latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 46 us, #4/4, CPU#1 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: sshd-1991 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: do_IRQ
#  => ended at:   do_IRQ
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
    sshd-1991    1d.h.    0us+: irq_enter <-do_IRQ
    sshd-1991    1d..1   46us : irq_exit <-do_IRQ
    sshd-1991    1d..1   47us+: trace_preempt_on <-do_IRQ
    sshd-1991    1d..1   52us : <stack trace>
 => sub_preempt_count
 => irq_exit
 => do_IRQ
 => ret_from_intr
```

Здесь видны некоторые отличия. Вытеснение было отключено во время прерывания (символ h) и восстановлено по завершении. Однако можно видеть, что прерывания были отключены при входе в раздел preempt off и на выходе из него (символ d). Неизвестно, были ли прерывания включены в это время или вскоре после него.

```
# tracer: preemptoff
#
# preemptoff latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 83 us, #241/241, CPU#1 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: bash-1994 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: wake_up_new_task
#  => ended at:   task_rq_unlock
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
```

```
#              |||| /     delay
#  cmd     pid  ||||| time  |   caller
#     \   /     |||||  \    |   /
    bash-1994    1d..1    0us : _raw_spin_lock_irqsave <-wake_up_new_task
    bash-1994    1d..1    0us : select_task_rq_fair <-select_task_rq
    bash-1994    1d..1    1us : __rcu_read_lock <-select_task_rq_fair
    bash-1994    1d..1    1us : source_load <-select_task_rq_fair
    bash-1994    1d..1    1us : source_load <-select_task_rq_fair
[...]
    bash-1994    1d..1   12us : irq_enter <-smp_apic_timer_interrupt
    bash-1994    1d..1   12us : rcu_irq_enter <-irq_enter
    bash-1994    1d..1   13us : add_preempt_count <-irq_enter
    bash-1994    1d.h1   13us : exit_idle <-smp_apic_timer_interrupt
    bash-1994    1d.h1   13us : hrtimer_interrupt <-smp_apic_timer_interrupt
    bash-1994    1d.h1   13us : _raw_spin_lock <-hrtimer_interrupt
    bash-1994    1d.h1   14us : add_preempt_count <-_raw_spin_lock
    bash-1994    1d.h2   14us : ktime_get_update_offsets <-hrtimer_interrupt
[...]
    bash-1994    1d.h1   35us : lapic_next_event <-clockevents_program_event
    bash-1994    1d.h1   35us : irq_exit <-smp_apic_timer_interrupt
    bash-1994    1d.h1   36us : sub_preempt_count <-irq_exit
    bash-1994    1d..2   36us : do_softirq <-irq_exit
    bash-1994    1d..2   36us : __do_softirq <-call_softirq
    bash-1994    1d..2   36us : __local_bh_disable <-__do_softirq
    bash-1994    1d.s2   37us : add_preempt_count <-_raw_spin_lock_irq
    bash-1994    1d.s3   38us : _raw_spin_unlock <-run_timer_softirq
    bash-1994    1d.s3   39us : sub_preempt_count <-_raw_spin_unlock
    bash-1994    1d.s2   39us : call_timer_fn <-run_timer_softirq
[...]
    bash-1994    1dNs2   81us : cpu_needs_another_gp <-rcu_process_callbacks
    bash-1994    1dNs2   82us : __local_bh_enable <-__do_softirq
    bash-1994    1dNs2   82us : sub_preempt_count <-__local_bh_enable
    bash-1994    1dN.2   82us : idle_cpu <-irq_exit
    bash-1994    1dN.2   83us : rcu_irq_exit <-irq_exit
    bash-1994    1dN.2   83us : sub_preempt_count <-irq_exit
    bash-1994    1.N.1   84us : _raw_spin_unlock_irqrestore <-task_rq_unlock
    bash-1994    1.N.1   84us+: trace_preempt_on <-task_rq_unlock
    bash-1994    1.N.1  104us : <stack trace>
 => sub_preempt_count
 => _raw_spin_unlock_irqrestore
 => task_rq_unlock
 => wake_up_new_task
 => do_fork
 => sys_clone
 => stub_clone
```

Выше приведен пример трассировки preemptoff с включенной опцией function-trace. Здесь видно, что прерывания были отключены не все время. Код irq_enter указывает вход в прерывание h. Перед этим отслеживаемые функции показывают, что они не находятся в прерывании, но из самих функций видно, что это не так.

Трассировщик preemptirqsoff

Полезно знать места, где прерывания или вытеснение были отключены на самое продолжительное время. Однако иногда нужно знать, когда прерывания и/или вытеснение были отключены. Рассмотрим пример кода.

```
local_irq_disable();
call_function_with_irqs_off();
preempt_disable();
call_function_with_irqs_and_preemption_off();
local_irq_enable();
call_function_with_preemption_off();
preempt_enable();
```

Трассировщик irqsoff записывает общую продолжительность вызовов call_function_with_irqs_off() и call_function_with_irqs_and_preemption_off(), preemptoff - общую продолжительность call_function_with_irqs_and_preemption_off() и call_function_with_preemption_off(). Но ни один из них не записывает время, в течение которого прерывания и/или вытеснение были отключены. Это общее время, которое мы не можем планировать. Для записи этого времени служит трассировщик preemptirqsoff.

Работа с этим трассировщиком похожа на действия с irqsoff и preemptoff.

```
# echo 0 > options/function-trace
# echo preemptirqsoff > current_tracer
# echo 1 > tracing_on
# echo 0 > tracing_max_latency
# ls -ltr
[...]
# echo 0 > tracing_on
# cat trace
# tracer: preemptirqsoff
#
# preemptirqsoff latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 100 us, #4/4, CPU#3 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: ls-2230 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: ata_scsi_queuecmd
#  => ended at:   ata_scsi_queuecmd
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
      ls-2230    3d...    0us+: _raw_spin_lock_irqsave <-ata_scsi_queuecmd
      ls-2230    3...1  100us : _raw_spin_unlock_irqrestore <-ata_scsi_queuecmd
      ls-2230    3...1  101us+: trace_preempt_on <-ata_scsi_queuecmd
      ls-2230    3...1  111us : <stack trace>
 => sub_preempt_count
 => _raw_spin_unlock_irqrestore
 => ata_scsi_queuecmd
 => scsi_dispatch_cmd
 => scsi_request_fn
 => __blk_run_queue_uncond
 => __blk_run_queue
```

```
  => blk_queue_bio
  => generic_make_request
  => submit_bio
  => submit_bh
  => ext3_bread
  => ext3_dir_bread
  => htree_dirblock_to_tree
  => ext3_htree_fill_tree
  => ext3_readdir
  => vfs_readdir
  => sys_getdents
  => system_call_fastpath
```

Вызов trace_hardirqs_off_thunk выполняется из сборки кода x86 когда прерывания запрещены в этой сборке. Без трассировки функций мы не узнаем, были ли прерывания запрещены в пределах точек вытеснения. Видно, что это начинается с включения вытеснения. Ниже показана трассировка с включенной опцией function-trace.

```
# tracer: preemptirqsoff
#
# preemptirqsoff latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 161 us, #339/339, CPU#3 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: ls-2269 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: schedule
#  => ended at:   mutex_unlock
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
kworker/-59      3...1    0us : __schedule <-schedule
kworker/-59      3d..1    0us : rcu_preempt_qs <-rcu_note_context_switch
kworker/-59      3d..1    1us : add_preempt_count <-_raw_spin_lock_irq
kworker/-59      3d..2    1us : deactivate_task <-__schedule
kworker/-59      3d..2    1us : dequeue_task <-deactivate_task
kworker/-59      3d..2    2us : update_rq_clock <-dequeue_task
kworker/-59      3d..2    2us : dequeue_task_fair <-dequeue_task
kworker/-59      3d..2    2us : update_curr <-dequeue_task_fair
kworker/-59      3d..2    2us : update_min_vruntime <-update_curr
kworker/-59      3d..2    3us : cpuacct_charge <-update_curr
kworker/-59      3d..2    3us : __rcu_read_lock <-cpuacct_charge
kworker/-59      3d..2    3us : __rcu_read_unlock <-cpuacct_charge
kworker/-59      3d..2    3us : update_cfs_rq_blocked_load <-dequeue_task_fair
kworker/-59      3d..2    4us : clear_buddies <-dequeue_task_fair
kworker/-59      3d..2    4us : account_entity_dequeue <-dequeue_task_fair
kworker/-59      3d..2    4us : update_min_vruntime <-dequeue_task_fair
kworker/-59      3d..2    4us : update_cfs_shares <-dequeue_task_fair
kworker/-59      3d..2    5us : hrtick_update <-dequeue_task_fair
kworker/-59      3d..2    5us : wq_worker_sleeping <-__schedule
kworker/-59      3d..2    5us : kthread_data <-wq_worker_sleeping
kworker/-59      3d..2    5us : put_prev_task_fair <-__schedule
kworker/-59      3d..2    6us : pick_next_task_fair <-pick_next_task
kworker/-59      3d..2    6us : clear_buddies <-pick_next_task_fair
kworker/-59      3d..2    6us : set_next_entity <-pick_next_task_fair
kworker/-59      3d..2    6us : update_stats_wait_end <-set_next_entity
      ls-2269    3d..2    7us : finish_task_switch <-__schedule
      ls-2269    3d..2    7us : _raw_spin_unlock_irq <-finish_task_switch
      ls-2269    3d..2    8us : do_IRQ <-ret_from_intr
      ls-2269    3d..2    8us : irq_enter <-do_IRQ
      ls-2269    3d..2    8us : rcu_irq_enter <-irq_enter
      ls-2269    3d..2    9us : add_preempt_count <-irq_enter
      ls-2269    3d.h2    9us : exit_idle <-do_IRQ
[...]
      ls-2269    3d.h3   20us : sub_preempt_count <-_raw_spin_unlock
      ls-2269    3d.h2   20us : irq_exit <-do_IRQ
      ls-2269    3d.h2   21us : sub_preempt_count <-irq_exit
      ls-2269    3d..3   21us : do_softirq <-irq_exit
      ls-2269    3d..3   21us : __do_softirq <-call_softirq
      ls-2269    3d..3   21us+: __local_bh_disable <-__do_softirq
      ls-2269    3d.s4   29us : sub_preempt_count <-_local_bh_enable_ip
      ls-2269    3d.s5   29us : sub_preempt_count <-_local_bh_enable_ip
      ls-2269    3d.s5   31us : do_IRQ <-ret_from_intr
      ls-2269    3d.s5   31us : irq_enter <-do_IRQ
      ls-2269    3d.s5   31us : rcu_irq_enter <-irq_enter
[...]
      ls-2269    3d.s5   31us : rcu_irq_enter <-irq_enter
      ls-2269    3d.s5   32us : add_preempt_count <-irq_enter
      ls-2269    3d.H5   32us : exit_idle <-do_IRQ
      ls-2269    3d.H5   32us : handle_irq <-do_IRQ
      ls-2269    3d.H5   32us : irq_to_desc <-handle_irq
      ls-2269    3d.H5   33us : handle_fasteoi_irq <-handle_irq
[...]
      ls-2269    3d.s5  158us : _raw_spin_unlock_irqrestore <-rtl8139_poll
      ls-2269    3d.s3  158us : net_rps_action_and_irq_enable.isra.65 <-net_rx_action
      ls-2269    3d.s3  159us : __local_bh_enable <-__do_softirq
      ls-2269    3d.s3  159us : sub_preempt_count <-__local_bh_enable
      ls-2269    3d..3  159us : idle_cpu <-irq_exit
      ls-2269    3d..3  159us : rcu_irq_exit <-irq_exit
      ls-2269    3d..3  160us : sub_preempt_count <-irq_exit
      ls-2269    3d...  161us : __mutex_unlock_slowpath <-mutex_unlock
      ls-2269    3d...  162us+: trace_hardirqs_on <-mutex_unlock
      ls-2269    3d...  186us : <stack trace>
  => __mutex_unlock_slowpath
  => mutex_unlock
  => process_output
  => n_tty_write
  => tty_write
  => vfs_write
  => sys_write
  => system_call_fastpath
```

Эта интересная трассировка, которая начинается с того, что процесс kworker работает и запланирован, но ls берет управление на себя. Как только ls освобождает rq и разрешает прерывания (но не вытеснение), сразу срабатывает прерывание. По завершении прерывания запускаются softirq, но во время работы softirq происходит другое прерывание, завершение которого внутри softirq указывается символом H.

Трассировщик wakeup

Одним из основных случаев, представляющих интерес для отслеживания, является время, требуемое для выполнения задачи, которая пробуждается для реальной работы. Это не задачи в реальном масштабе времени (RT), а обычные задачи. Но их трассировка не менее интересна.

Ниже приведена трассировка без отслеживания функций.

```
# echo 0 > options/function-trace
# echo wakeup > current_tracer
# echo 1 > tracing_on
# echo 0 > tracing_max_latency
# chrt -f 5 sleep 1
# echo 0 > tracing_on
# cat trace
# tracer: wakeup
#
# wakeup latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 15 us, #4/4, CPU#3 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: kworker/3:1H-312 (uid:0 nice:-20 policy:0 rt_prio:0)
#    -----------------
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
  <idle>-0       3dNs7    0us :      0:120:R   + [003]   312:100:R kworker/3:1H
  <idle>-0       3dNs7    1us+: ttwu_do_activate.constprop.87 <-try_to_wake_up
  <idle>-0       3d..3   15us : __schedule <-schedule
  <idle>-0       3d..3   15us :      0:120:R ==> [003]   312:100:R kworker/3:1H
```

Трассировщик отслеживат лишь задачи с наиболее высоким приоритетом в системе, не трассируя обычные ситуации. Можно видеть, что для задачи kworker с приоритетом nice = of -20 (не очень высокий) прошло лишь 15 мксек между пробуждением и началом работы.

Задачи, не работающие в режиме реального времени (RT), не так интересны, по сравнению с задачами RT.

Трассировщик wakeup_rt

В среде RT очень важно знать, какое время проходит между пробуждением задачи с наивысшим приоритетом и началом ее выполнения. Это время называют также задержкой планирования (schedule latency). Подчеркнем, что речь здесь идет только о задачах RT. Важно также понимать задержку планирования для задач не-RT, но для них лучше средняя задержка планирования. Для таких измерений лучше подходят такие инструменты, как LatencyTop.

Для сред RT интересна задержка в худшей ситуации. Это наибольшая, а не средняя задержка. У нас может быть очень быстрый планировщик, который будет редко приводить к значительным задержкам, но он может быть недостаточно хорош для задач RT. Трассировщик wakeup_rt был разработан для определения худшего варианта пробуждения задач RT. Другие (не-RT) задачи не отслеживаются, поскольку трассировщик записывает один наихудший случай, а отслеживание задач не-RT, дающее непредсказуемый результат, будет переписывать наихудшую задержку для задач RT (это легко увидеть при запуске трассировщика wakeup).

Поскольку этот трассировщик работает только с задачами RT, воспользуемся для него вместо привычной команды ls командой sleep 1 в chrt, где меняется приоритет задаси.

```
# echo 0 > options/function-trace
# echo wakeup_rt > current_tracer
# echo 1 > tracing_on
# echo 0 > tracing_max_latency
# chrt -f 5 sleep 1
# echo 0 > tracing_on
# cat trace
# tracer: wakeup
#
# tracer: wakeup_rt
#
# wakeup_rt latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 5 us, #4/4, CPU#3 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: sleep-2389 (uid:0 nice:0 policy:1 rt_prio:5)
#    -----------------
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
  <idle>-0       3d.h4    0us :      0:120:R   + [003]  2389: 94:R sleep
  <idle>-0       3d.h4    1us+: ttwu_do_activate.constprop.87 <-try_to_wake_up
  <idle>-0       3d..3    5us : __schedule <-schedule
  <idle>-0       3d..3    5us :      0:120:R ==> [003]  2389: 94:R sleep
```

Запустив это в режиме ожидания, видим, что для переключения задачи потребовалось лишь 5 мксек. Отметим, что трассировка была остановлена к моменту, когда записываемая задача была запланирована, поскольку точка трассировки в планировании расположена перед фактическим переключателем (switch). Это можно изменить, добавив новый маркер в конце планировщика.

Отметим, что записываемая задача «спит» (sleep) с PID = 2389 и имеет rt_prio = 5. Этот приоритет относится к пользовательскому пространству, а не к ядру. Для SCHED_FIFO используется правило 1, для SCHED_RR - 2.

Отметим, что данные трассировки показывают внутренний приоритет (99 - rt_prio).

```
  <idle>-0       3d..3    5us :      0:120:R ==> [003]  2389: 94:R sleep
```

Запись 0:120:R означает бездействие с приоритетом nice = 0 (120 — 120), в рабочем состоянии (R). Спящая задача запланирована с 2389: 94:R. Т. е. приоритет является приоритетом ядра rt_prio (99 - 5 = 94) и задача также выполняется.

Повторим это с chrt -r 5 и установленной опцией function-trace.

```
echo 1 > options/function-trace

# tracer: wakeup_rt
#
# wakeup_rt latency trace v1.1.5 on 3.8.0-test+
# --------------------------------------------------------------------
# latency: 29 us, #85/85, CPU#3 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
#    -----------------
#    | task: sleep-2448 (uid:0 nice:0 policy:1 rt_prio:5)
#    -----------------
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
  <idle>-0       3d.h4    1us+:      0:120:R   + [003]  2448: 94:R sleep
  <idle>-0       3d.h4    2us : ttwu_do_activate.constprop.87 <-try_to_wake_up
  <idle>-0       3d.h3    3us : check_preempt_curr <-ttwu_do_wakeup
  <idle>-0       3d.h3    3us : resched_curr <-check_preempt_curr
  <idle>-0       3dNh3    4us : task_woken_rt <-ttwu_do_wakeup
  <idle>-0       3dNh3    4us : _raw_spin_unlock <-try_to_wake_up
  <idle>-0       3dNh3    4us : sub_preempt_count <-_raw_spin_unlock
  <idle>-0       3dNh2    5us : ttwu_stat <-try_to_wake_up
  <idle>-0       3dNh2    5us : _raw_spin_unlock_irqrestore <-try_to_wake_up
  <idle>-0       3dNh2    6us : sub_preempt_count <-_raw_spin_unlock_irqrestore
  <idle>-0       3dNh1    6us : _raw_spin_lock <-__run_hrtimer
  <idle>-0       3dNh1    6us : add_preempt_count <-_raw_spin_lock
  <idle>-0       3dNh2    7us : _raw_spin_unlock <-hrtimer_interrupt
  <idle>-0       3dNh2    7us : sub_preempt_count <-_raw_spin_unlock
  <idle>-0       3dNh1    7us : tick_program_event <-hrtimer_interrupt
  <idle>-0       3dNh1    7us : clockevents_program_event <-tick_program_event
  <idle>-0       3dNh1    8us : ktime_get <-clockevents_program_event
  <idle>-0       3dNh1    8us : lapic_next_event <-clockevents_program_event
  <idle>-0       3dNh1    8us : irq_exit <-smp_apic_timer_interrupt
  <idle>-0       3dNh1    9us : sub_preempt_count <-irq_exit
  <idle>-0       3dN.2    9us : idle_cpu <-irq_exit
  <idle>-0       3dN.2    9us : rcu_irq_exit <-irq_exit
  <idle>-0       3dN.2   10us : rcu_eqs_enter_common.isra.45 <-rcu_irq_exit
  <idle>-0       3dN.2   10us : sub_preempt_count <-irq_exit
  <idle>-0       3.N.1   11us : rcu_idle_exit <-cpu_idle
  <idle>-0       3dN.1   11us : rcu_eqs_exit_common.isra.43 <-rcu_idle_exit
  <idle>-0       3.N.1   11us : tick_nohz_idle_exit <-cpu_idle
  <idle>-0       3dN.1   12us : menu_hrtimer_cancel <-tick_nohz_idle_exit
  <idle>-0       3dN.1   12us : ktime_get <-tick_nohz_idle_exit
  <idle>-0       3dN.1   12us : tick_do_update_jiffies64 <-tick_nohz_idle_exit
  <idle>-0       3dN.1   13us : cpu_load_update_nohz <-tick_nohz_idle_exit
  <idle>-0       3dN.1   13us : _raw_spin_lock <-cpu_load_update_nohz
  <idle>-0       3dN.1   13us : add_preempt_count <-_raw_spin_lock
  <idle>-0       3dN.2   13us : __cpu_load_update <-cpu_load_update_nohz
  <idle>-0       3dN.2   14us : sched_avg_update <-__cpu_load_update
  <idle>-0       3dN.2   14us : _raw_spin_unlock <-cpu_load_update_nohz
  <idle>-0       3dN.2   14us : sub_preempt_count <-_raw_spin_unlock
  <idle>-0       3dN.1   15us : calc_load_nohz_stop <-tick_nohz_idle_exit
  <idle>-0       3dN.1   15us : touch_softlockup_watchdog <-tick_nohz_idle_exit
  <idle>-0       3dN.1   15us : hrtimer_cancel <-tick_nohz_idle_exit
  <idle>-0       3dN.1   15us : hrtimer_try_to_cancel <-hrtimer_cancel
  <idle>-0       3dN.1   16us : lock_hrtimer_base.isra.18 <-hrtimer_try_to_cancel
  <idle>-0       3dN.1   16us : _raw_spin_lock_irqsave <-lock_hrtimer_base.isra.18
  <idle>-0       3dN.1   16us : add_preempt_count <-_raw_spin_lock_irqsave
  <idle>-0       3dN.2   17us : __remove_hrtimer <-remove_hrtimer.part.16
  <idle>-0       3dN.2   17us : hrtimer_force_reprogram <-__remove_hrtimer
  <idle>-0       3dN.2   17us : tick_program_event <-hrtimer_force_reprogram
  <idle>-0       3dN.2   18us : clockevents_program_event <-tick_program_event
  <idle>-0       3dN.2   18us : ktime_get <-clockevents_program_event
  <idle>-0       3dN.2   18us : lapic_next_event <-clockevents_program_event
  <idle>-0       3dN.2   19us : _raw_spin_unlock_irqrestore <-hrtimer_try_to_cancel
  <idle>-0       3dN.2   19us : sub_preempt_count <-_raw_spin_unlock_irqrestore
  <idle>-0       3dN.1   19us : hrtimer_forward <-tick_nohz_idle_exit
  <idle>-0       3dN.1   20us : ktime_add_safe <-hrtimer_forward
  <idle>-0       3dN.1   20us : ktime_add_safe <-hrtimer_forward
  <idle>-0       3dN.1   20us : hrtimer_start_range_ns <-hrtimer_start_expires.constprop.11
  <idle>-0       3dN.1   20us : __hrtimer_start_range_ns <-hrtimer_start_range_ns
  <idle>-0       3dN.1   21us : lock_hrtimer_base.isra.18 <-__hrtimer_start_range_ns
  <idle>-0       3dN.1   21us : _raw_spin_lock_irqsave <-lock_hrtimer_base.isra.18
  <idle>-0       3dN.1   21us : add_preempt_count <-_raw_spin_lock_irqsave
  <idle>-0       3dN.2   22us : ktime_add_safe <-__hrtimer_start_range_ns
  <idle>-0       3dN.2   22us : enqueue_hrtimer <-__hrtimer_start_range_ns
  <idle>-0       3dN.2   22us : tick_program_event <-__hrtimer_start_range_ns
  <idle>-0       3dN.2   23us : clockevents_program_event <-tick_program_event
  <idle>-0       3dN.2   23us : ktime_get <-clockevents_program_event
  <idle>-0       3dN.2   23us : lapic_next_event <-clockevents_program_event
  <idle>-0       3dN.2   24us : _raw_spin_unlock_irqrestore <-__hrtimer_start_range_ns
  <idle>-0       3dN.2   24us : sub_preempt_count <-_raw_spin_unlock_irqrestore
  <idle>-0       3dN.1   24us : account_idle_ticks <-tick_nohz_idle_exit
  <idle>-0       3dN.1   24us : account_idle_time <-account_idle_ticks
  <idle>-0       3.N.1   25us : sub_preempt_count <-cpu_idle
  <idle>-0       3.N..   25us : schedule <-cpu_idle
  <idle>-0       3.N..   25us : __schedule <-preempt_schedule
  <idle>-0       3.N..   26us : add_preempt_count <-__schedule
  <idle>-0       3.N.1   26us : rcu_note_context_switch <-__schedule
  <idle>-0       3.N.1   26us : rcu_sched_qs <-rcu_note_context_switch
  <idle>-0       3dN.1   27us : rcu_preempt_qs <-rcu_note_context_switch
  <idle>-0       3.N.1   27us : _raw_spin_lock_irq <-__schedule
  <idle>-0       3dN.1   27us : add_preempt_count <-_raw_spin_lock_irq
  <idle>-0       3dN.2   28us : put_prev_task_idle <-__schedule
  <idle>-0       3dN.2   28us : pick_next_task_stop <-pick_next_task
  <idle>-0       3dN.2   28us : pick_next_task_rt <-pick_next_task
  <idle>-0       3dN.2   29us : dequeue_pushable_task <-pick_next_task_rt
  <idle>-0       3d..3   29us : __schedule <-preempt_schedule
  <idle>-0       3d..3   30us :      0:120:R ==> [003]  2448: 94:R sleep
```

Трек даже при отслеживании функций не слишком велик и приведен здесь полностью.

Прерывание произошло в момент бездействия системы. Где-то перед вызовом task_woken_rt() был установлен флаг NEED_RESCHED, что показано первым включением символа N.

Трассировка задержки и события

Трассировка функций может вносить значительные задержки, но без нее трудно установить причины задержки. Золотой серединой является включение трассировки событий, как показано ниже.

```
  # echo 0 > options/function-trace
  # echo wakeup_rt > current_tracer
  # echo 1 > events/enable
  # echo 1 > tracing_on
  # echo 0 > tracing_max_latency
  # chrt -f 5 sleep 1
  # echo 0 > tracing_on
  # cat trace
  # tracer: wakeup_rt
  #
  # wakeup_rt latency trace v1.1.5 on 3.8.0-test+
  # --------------------------------------------------------------------
  # latency: 6 us, #12/12, CPU#2 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:4)
  #    -----------------
  #    | task: sleep-5882 (uid:0 nice:0 policy:1 rt_prio:5)
  #    -----------------
  #
  #                  _------=> CPU#
  #                 / _-----=> irqs-off
  #                | / _----=> need-resched
  #                || / _---=> hardirq/softirq
  #                ||| / _--=> preempt-depth
  #                |||| /     delay
  #  cmd     pid   ||||| time  |   caller
  #     \   /      |||||  \    |   /
    <idle>-0       2d.h4    0us :      0:120:R   + [002]  5882: 94:R sleep
    <idle>-0       2d.h4    0us : ttwu_do_activate.constprop.87 <-try_to_wake_up
    <idle>-0       2d.h4    1us : sched_wakeup: comm=sleep pid=5882 prio=94 success=1 target_cpu=002
    <idle>-0       2dNh2    1us : hrtimer_expire_exit: hrtimer=ffff88007796feb8
    <idle>-0       2.N.2    2us : power_end: cpu_id=2
    <idle>-0       2.N.2    3us : cpu_idle: state=4294967295 cpu_id=2
    <idle>-0       2dN.3    4us : hrtimer_cancel: hrtimer=ffff88007d50d5e0
    <idle>-0       2dN.3    4us : hrtimer_start: hrtimer=ffff88007d50d5e0 function=tick_sched_timer expires=34311211000000
softexpires=34311211000000
    <idle>-0       2.N.2    5us : rcu_utilization: Start context switch
    <idle>-0       2.N.2    5us : rcu_utilization: End context switch
    <idle>-0       2d..3    6us : __schedule <-schedule
    <idle>-0       2d..3    6us :      0:120:R ==> [002]  5882: 94:R sleep
```

Определение аппаратной задержки

Для определения задержек в оборудовании служит трассировщик hwlat. Следует отметить, что этот трассировщик влияет на производительность всей системы и может полностью загружать CPU с запретом прерываний.

```
  # echo hwlat > current_tracer
  # sleep 100
  # cat trace
  # tracer: hwlat
  #
  #                              _-----=> irqs-off
  #                             / _----=> need-resched
  #                            | / _---=> hardirq/softirq
  #                            || / _--=> preempt-depth
  #                            ||| /     delay
  #           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
  #              | |       |   ||||       |         |
            <...>-3638  [001] d... 19452.055471: #1     inner/outer(us):   12/14    ts:1499801089.066141940
            <...>-3638  [003] d... 19454.071354: #2     inner/outer(us):   11/9     ts:1499801091.082164365
            <...>-3638  [002] dn.. 19461.126852: #3     inner/outer(us):   12/9     ts:1499801098.138150062
            <...>-3638  [001] d... 19488.340960: #4     inner/outer(us):    8/12    ts:1499801125.354139633
            <...>-3638  [003] d... 19494.388553: #5     inner/outer(us):    8/12    ts:1499801131.402150961
            <...>-3638  [003] d... 19501.283419: #6     inner/outer(us):    0/12    ts:1499801138.297435289 nmi-total:4 nmi-count:1
```

Приведенный выше вывод содержит заголовок с описанием формата. Для всех событий прерывания запрещены (d). Вывод в колонке FUNCTION описан ниже.

#1

Число записанных событий, превышающих tracing_threshold (см. ниже).

inner/outer(us): 12/14

Два значения, показывающие внутреннюю (inner) и внешнюю (outer) задержку. Тест выполняется в цикле, проверяя временную метку дважды. Задержка между двумя метками одного цикла называется внутренней, а задержка между меткой предыдущего и текущего циклов - внешней.

ts:1499801089.066141940

Метка абсолютного времени события.

nmi-total:4 nmi-count:1

Для поддерживающей немаскируемые прерывания архитектуры при NMI в процессе тестирования время, проведенное в NMI, указывается в поле nmi-total (мксек).

Если архитектура поддерживает NMI, поле nmi-count будет показывать число NMI в процессе тестирования.

Файлы hwlat

tracing_thresh

Если в этом файле указано значение 0, автоматически устанавливается значение 10 (мксек), определяющее порог задержки, которая вызывает запись трассировки в кольцевой буфер.

Отметим, что по завершении трассировщика hwlat (записи другого трассировщика в current_tracer) в файле восстанавливается исходное значение tracing_threshold.

hwlat_detector/width

Продолжительность работы теста с отключенными прерываниями.

hwlat_detector/window

Продолжительность окна тестирования - тест выполнялся в течение width мксек в течение window мксек.

tracing_cpumask

При запуске теста создается поток (thread) ядра для работы теста. Этот поток будет передаваться между CPU, указанными в tracing_cpumask каждый интервал (окно - window). Для работы теста на определенных CPU в этом файле устанавливается нужная маска.

Трассировщик function

Этот трассировщик отслеживает функции. Включить трассировщик можно через файловую систему debugfs. Следует убедиться, что установлена переменная (файл) ftrace_enabled, поскольку в противном случае трассировка не будет делать ничего (см. ftrace_enabled ниже).

Ниже приведен пример команд и вывода при трассировке функций.

```
# sysctl kernel.ftrace_enabled=1
# echo function > current_tracer
# echo 1 > tracing_on
# usleep 1
# echo 0 > tracing_on
# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 24799/24799   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
           bash-1994  [002] ....  3082.063030: mutex_unlock <-rb_simple_write
           bash-1994  [002] ....  3082.063031: __mutex_unlock_slowpath <-mutex_unlock
           bash-1994  [002] ....  3082.063031: __fsnotify_parent <-fsnotify_modify
           bash-1994  [002] ....  3082.063032: fsnotify <-fsnotify_modify
           bash-1994  [002] ....  3082.063032: __srcu_read_lock <-fsnotify
           bash-1994  [002] ....  3082.063032: add_preempt_count <-__srcu_read_lock
           bash-1994  [002] ...1  3082.063032: sub_preempt_count <-__srcu_read_lock
           bash-1994  [002] ....  3082.063033: __srcu_read_unlock <-fsnotify
```

Примечание. Трассировщик функций использует кольцевые буферы для хранения показанных выше записей и новые данные могут переписывать более старые. Иногда использования команды echo не достаточно для прекращения трассировки, поскольку данные могут быть перезаписаны. По этой причине иногда лучше отключать трассировку из программы. Это позволит остановить трассировку в нужной точке. Для выключения трассировки непосредственно из программы на языке C можно воспользоваться кодом, подобным приведенному ниже.

```
int trace_fd;
[...]
int main(int argc, char *argv[]) {
          [...]
          trace_fd = open(tracing_file("tracing_on"), O_WRONLY);
          [...]
          if (condition_hit()) {
                    write(trace_fd, "0", 1);
          }
          [...]
}
```

Трассировка одного потока

Путем записи нужного значения в файл set_ftrace_pid можно ограничиться отслеживанием одного потока (thread).

```
# cat set_ftrace_pid
no pid
# echo 3111 > set_ftrace_pid
# cat set_ftrace_pid
3111
# echo function > current_tracer
# cat trace | head
# tracer: function
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
    yum-updatesd-3111  [003]  1637.254676: finish_task_switch <-thread_return
    yum-updatesd-3111  [003]  1637.254681: hrtimer_cancel <-schedule_hrtimeout_range
    yum-updatesd-3111  [003]  1637.254682: hrtimer_try_to_cancel <-hrtimer_cancel
    yum-updatesd-3111  [003]  1637.254683: lock_hrtimer_base <-hrtimer_try_to_cancel
    yum-updatesd-3111  [003]  1637.254685: fget_light <-do_sys_poll
    yum-updatesd-3111  [003]  1637.254686: pipe_poll <-do_sys_poll
# echo > set_ftrace_pid
# cat trace |head
# tracer: function
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
##### CPU 3 buffer started ####
    yum-updatesd-3111  [003]  1701.957688: free_poll_entry <-poll_freewait
    yum-updatesd-3111  [003]  1701.957689: remove_wait_queue <-free_poll_entry
    yum-updatesd-3111  [003]  1701.957691: fput <-free_poll_entry
    yum-updatesd-3111  [003]  1701.957692: audit_syscall_exit <-sysret_audit
    yum-updatesd-3111  [003]  1701.957693: path_put <-audit_syscall_exit
```

Если нужно отслеживать функцию при выполнении, можно воспользоваться чем-то вроде приведенной программы.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define _STR(x) #x
#define STR(x) _STR(x)
#define MAX_PATH 256

const char *find_tracefs(void)
{
       static char tracefs[MAX_PATH+1];
       static int tracefs_found;
       char type[100];
```

```
        FILE *fp;

        if (tracefs_found)
                return tracefs;

        if ((fp = fopen("/proc/mounts","r")) == NULL) {
                perror("/proc/mounts");
                return NULL;
        }

        while (fscanf(fp, "%*s %"
                      STR(MAX_PATH)
                      "s %99s %*s %*d %*d\n",
                      tracefs, type) == 2) {
                if (strcmp(type, "tracefs") == 0)
                        break;
        }
        fclose(fp);

        if (strcmp(type, "tracefs") != 0) {
                fprintf(stderr, "tracefs not mounted");
                return NULL;
        }

        strcat(tracefs, "/tracing/");
        tracefs_found = 1;

        return tracefs;
}

const char *tracing_file(const char *file_name)
{
        static char trace_file[MAX_PATH+1];
        snprintf(trace_file, MAX_PATH, "%s/%s", find_tracefs(), file_name);
        return trace_file;
}

int main (int argc, char **argv)
{
          if (argc < 1)
                  exit(-1);

          if (fork() > 0) {
                  int fd, ffd;
                  char line[64];
                  int s;

                  ffd = open(tracing_file("current_tracer"), O_WRONLY);
                  if (ffd < 0)
                          exit(-1);
                  write(ffd, "nop", 3);

                  fd = open(tracing_file("set_ftrace_pid"), O_WRONLY);
                  s = sprintf(line, "%d\n", getpid());
                  write(fd, line, s);

                  write(ffd, "function", 8);

                  close(fd);
                  close(ffd);

                  execvp(argv[1], argv+1);
          }

          return 0;
}
```

Подойдет также простой сценарий, показанный ниже.

```
#!/bin/bash

tracefs=`sed -ne 's/^tracefs \(.*\) tracefs.*/\1/p' /proc/mounts`
echo nop > $tracefs/tracing/current_tracer
echo 0 > $tracefs/tracing/tracing_on
echo $$ > $tracefs/tracing/set_ftrace_pid
echo function > $tracefs/tracing/current_tracer
echo 1 > $tracefs/tracing/tracing_on
exec "$@"
```

Трассировщик function_graph

Этот трассировщик похож на function, но проверяет функции на входе и выходе. Это делается с помощью динамически выделяемого стека адресов возврата в каждой task_struct. На входе функции трассировщик переписывает адрес возврата каждой отслеживаемой функции для установки специального «зонда». Исходный адрес возврата сохраняется в стеке возврата структуры task_struct.

Зондирование на входе и выходе функции позволяет определить время ее выполнения и получить надежный стек вызовов для создания графа вызовов функций.

Этот трассировщик полезен в нескольких ситуациях:

- поиск причин странного поведения ядра;
- обнаружение непонятных задержек;
- быстрый поиск пути выполнение определенной функции;
- исследование происходящего внутри ядра.

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |

 0)               |  sys_open() {
 0)               |    do_sys_open() {
 0)               |      getname() {
 0)               |        kmem_cache_alloc() {
 0)   1.382 us    |          __might_sleep();
 0)   2.478 us    |        }
 0)               |        strncpy_from_user() {
 0)               |          might_fault() {
 0)   1.389 us    |            __might_sleep();
 0)   2.553 us    |          }
 0)   3.807 us    |        }
 0)   7.876 us    |      }
 0)               |      alloc_fd() {
 0)   0.668 us    |        _spin_lock();
 0)   0.570 us    |        expand_files();
 0)   0.586 us    |        _spin_unlock();
```

Здесь имеется несколько колонок, которые можно динамически включать и выключать. Можно использовать любую подходящую комбинацию опций.

- Номер процессора, на котором выполняется функция, по умолчанию выводится. Иногда может потребоваться отслеживание лишь одного процессора (см. файл tracing_cpu_mask) или отслеживать функции без привязки к процессорам.
  - Скрыть номер процессора - echo nofuncgraph-cpu > trace_options.
  - Показывать номер процессора - echo funcgraph-cpu > trace_options.
- Продолжительность выполнения функции указывается после закрывающей (фигурной) скобки функции или в одной строке с функцией, если эта функция не делает других вызовов. По умолчнию отображение продолжительности включено.
  - Скрыть продолжительность вызова - echo nofuncgraph-duration > trace_options.
  - Показывать продолжительность вызова - echo funcgraph-duration > trace_options.
- Поле служебных данных (overhead) указывается перед полем продолжительности в случае достижения порога длительности.
  - Скрыть служебные данные - echo nofuncgraph-overhead > trace_options.
  - Показывать служебные данные - echo funcgraph-overhead > trace_options

```
3) # 1837.709 us |          } /* __switch_to */
3)               |          finish_task_switch() {
3)   0.313 us    |            _raw_spin_unlock_irq();
3)   3.177 us    |          }
3) # 1889.063 us |        } /* __schedule */
3) ! 140.417 us  |      } /* __schedule */
3) # 2034.948 us |    } /* schedule */
3) * 33998.59 us |  } /* schedule_preempt_disabled */

[...]

1)   0.260 us    |              msecs_to_jiffies();
1)   0.313 us    |              __rcu_read_unlock();
1) + 61.770 us   |            }
1) + 64.479 us   |          }
1)   0.313 us    |          rcu_bh_qs();
1)   0.313 us    |          __local_bh_enable();
1) ! 217.240 us  |        }
1)   0.365 us    |        idle_cpu();
1)               |        rcu_irq_exit() {
1)   0.417 us    |          rcu_eqs_enter_common.isra.47();
1)   3.125 us    |        }
1) ! 227.812 us  |      }
1) ! 457.395 us  |    }
1) @ 119760.2 us |  }

[...]

2)               |    handle_IPI() {
1)   6.979 us    |                }
2)   0.417 us    |      scheduler_ipi();
1)   9.791 us    |              }
1) + 12.917 us   |            }
2)   3.490 us    |    }
1) + 15.729 us   |          }
1) + 18.542 us   |        }
2) $ 3594274 us  |  }
```

Маркеры продолжительности

\+ - больше 10 мксек.

! - больше 100 мксек.

- больше 1000 мксек.

* - больше 10 мсек.

@ - больше 100 мсек.

$ - больше 1 сек.

- Поле task/pid показывает строку команды и pid процесса, в котором функция выполняется. По умолчанию это поле не выводится.
  - Скрыть поле - echo nofuncgraph-proc > trace_options.
  - Показывать поле - echo funcgraph-proc > trace_options.

```
# tracer: function_graph
#
# CPU  TASK/PID        DURATION                  FUNCTION CALLS
# |     |    |           |   |                     |   |   |   |
0)    sh-4802     |               |                  d_free() {
0)    sh-4802     |               |                    call_rcu() {
0)    sh-4802     |               |                      __call_rcu() {
0)    sh-4802     |   0.616 us    |                        rcu_process_gp_end();
0)    sh-4802     |   0.586 us    |                        check_for_new_grace_period();
0)    sh-4802     |   2.899 us    |                      }
0)    sh-4802     |   4.040 us    |                    }
0)    sh-4802     |   5.151 us    |                  }
0)    sh-4802     | + 49.370 us   |                }
```

- Поле «абсолютного» времени содержит временную метку системных часов, отсчитываемую от момента загрузки системы. Время указывается на входе и выходе функции.
  - Скрыть время - echo nofuncgraph-abstime > trace_options
  - Показывать время - echo funcgraph-abstime > trace_options

```
#
#      TIME       CPU  DURATION                  FUNCTION CALLS
#       |         |     |   |                     |   |   |   |
360.774522 |   1)   0.541 us    |                                  }
360.774522 |   1)   4.663 us    |                                }
360.774523 |   1)   0.541 us    |                                __wake_up_bit();
360.774524 |   1)   6.796 us    |                              }
360.774524 |   1)   7.952 us    |                            }
360.774525 |   1)   9.063 us    |                          }
360.774525 |   1)   0.615 us    |                          journal_mark_dirty();
360.774527 |   1)   0.578 us    |                          __brelse();
360.774528 |   1)               |                          reiserfs_prepare_for_journal() {
360.774528 |   1)               |                            unlock_buffer() {
360.774529 |   1)               |                              wake_up_bit() {
360.774529 |   1)               |                                bit_waitqueue() {
360.774530 |   1)   0.594 us    |                                  __phys_addr();
```

Имя функции, начало которой не попадает в буфер трассировки выводится после закрывающей скобки для функции. Вывод этого имени можно включить для упрощения поиска продолжительности функций с помощью grep. По умолчанию вывод отключен

- Скрыть имя - echo nofuncgraph-tail > trace_options.
- Показывать имя - echo funcgraph-tail > trace_options.

Пример без вывода имен (nofuncgraph-tail), как принято по умолчанию.

```
0)               |      putname() {
0)               |        kmem_cache_free() {
0)   0.518 us    |          __phys_addr();
0)   1.757 us    |        }
0)   2.861 us    |      }
```

Пример с выводом имен (funcgraph-tail)

```
0)               |      putname() {
0)               |        kmem_cache_free() {
0)   0.518 us    |          __phys_addr();
0)   1.757 us    |        } /* kmem_cache_free() */
0)   2.861 us    |      } /* putname() */
```

Можно добавить комментарии к конкретным функциям с помощью trace_printk(). Например, для включения комментариев в функцию __might_sleep() достаточно включить файл <linux/ftrace.h> и вызвать trace_printk() внутри __might_sleep(). Например,

```
trace_printk("I'm a comment!\n")
```

будет давать вывод

```
1)               |            __might_sleep() {
1)               |              /* I'm a comment! */
1)   1.449 us    |            }
```

Другие варианты применения этого трассировщика описаны в следующем разделе.

Динамическая трассировка ftrace

Если установлена опция ядра CONFIG_DYNAMIC_FTRACE, система будет работать почти без связанных с трассировкой издержек при отключенной трассировке функций. Это работает за счет вызова функции mcount (помещается в начале каждой функции ядра с помощью опции -pg компилятора gcc[1]), указывающей простой возврат.

При компиляции каждый объект файла C запускается через программу recordmcount (находится в каталоге scripts), котоая анализирует заголовки ELF в объекте C для поиска всех вызовов в разделе .text, вызывающих функцию mcount[2].

Отметим, что отслеживаются не все разделы - трассировка может быть отключена notrace или заблокирована иным путем и все встроенные функции не будут отслеживаться. Для просмотра списка функций, которые могут трассироваться, следует обратиться к файлу available_filter_functions.

Создается раздел __mcount_loc, в котором содержатся ссылки на все вызовы mcount/fentry в разделе .text. Программа recordmcount заново связывает этот раздел с исходными объектами. На этапе финальной компоновки ядра все эти ссылки будут добавлены в одну таблицу.

При загрузке (до инициализации SMP) динамический код ftrace сканирует эту таблицу и заменяет все найденные ссылки на nop, а также записывает местоположения, добавленные в список available_filter_functions. Модули обрабатываются по мере их загрузки до начала выполнения. При выгрузке модуля реализованные в нем функции удаляются из списка функций ftrace. Эту процедуру автоматически выполняет код выгрузки модулей и автор модуля не должен беспокоиться об этом.

[1]Включение FTRACE будет добавлять опции -pg при компиляции ядра
[2]Начиная с gcc версии 4.6 для процессоров x86 можно добавить опцию -mfentry, которая будет вызывать __fentry__ взамен mcount до создания кадра стека.

При включенной трассировке процесс изменения точек трассировки функций зависит от архитектуры. Старый метод использовал kstop_machine для предотвращения «гонок» CPU, выполняющих измененный код (что может заставлять CPU делать нежелательные вещи, особенно в тех случаях, когда измененный код пересекает границу кэше или страницы) и операции nop обратно помещались в вызовы. Но сейчас вместо mcount (сейчас это просто функция-заглушка) используются вызовы инфраструктуры ftrace.

Новый метод изменения точек трассировки функций заключается в размещении точки остановки (breakpoint) в изменяемом месте, синхронизации всех CPU, и изменении остальной части инструкции, не охваченной точкой остановки. CPU синхронизируются еще раз и точка установки удаляется из окончательной версии.

Некоторым вариантам архитектуры не нужно заботиться о синхронизации, они просто могут поместить новый код взамен прежнего без возникновения проблем с другими CPU, работающими в это же время.

Одним из побочных эффектов записи функций, которые будут отслеживаться, является то, что можно выбрать функции, которые мы хотим трассировать, и функции, для которых хотим, чтобы вызовы mcount оставались nop.

Используется два файла, один из которых служит для разрешения, другой для запрета трассировки заданных функций

```
set_ftrace_filter
```

и

```
set_ftrace_notrace
```

Список функций, которые можно включить в эти файлы содержится в файле available_filter_functions.

```
# cat available_filter_functions
put_prev_task_idle
kmem_cache_create
pick_next_task_rt
get_online_cpus
pick_next_task_fair
mutex_lock
[...]
```

Если интересно отслеживать лишь sys_nanosleep и hrtimer_interrupt, можно воспользоваться приведенными ниже командами.

```
# echo sys_nanosleep hrtimer_interrupt > set_ftrace_filter
# echo function > current_tracer
# echo 1 > tracing_on
# usleep 1
# echo 0 > tracing_on
# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 5/5   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          usleep-2665  [001] ....  4186.475355: sys_nanosleep <-system_call_fastpath
          <idle>-0     [001] d.h1  4186.475409: hrtimer_interrupt <-smp_apic_timer_interrupt
          usleep-2665  [001] d.h1  4186.475426: hrtimer_interrupt <-smp_apic_timer_interrupt
          <idle>-0     [003] d.h1  4186.475426: hrtimer_interrupt <-smp_apic_timer_interrupt
          <idle>-0     [002] d.h1  4186.475427: hrtimer_interrupt <-smp_apic_timer_interrupt
```

Для просмотра функций, которые будут отслеживаться, можно воспользоваться командой

```
# cat set_ftrace_filter
hrtimer_interrupt
sys_nanosleep
```

Иногда явного указания имен всех функций не достаточно, поэтому фильтры поддерживают шаблоны сопоставлений (см. glob).

match*
: соответствует всем функциям, начинающимся с match.

****match***
: соответствует всем функциям, заканчивающимся match.

****match****
: соответствует всем функциям, содержащим match в имени.

match1*match2
: соответствует всем функциям, начинающимся с match1 и заканчивающимися match2.

Примечание. Шаблоны сопоставления лучше указывать в кавычках, поскольку в противном случае интерпретатор команд может счесть параметры именами файлов в локальном каталоге.

Например,

```
# echo 'hrtimer_*' > set_ftrace_filter
```

Приведет к выводу, показанному ниже

```
# tracer: function
#
# entries-in-buffer/entries-written: 897/897   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          <idle>-0     [003] dN.1  4228.547803: hrtimer_cancel <-tick_nohz_idle_exit
          <idle>-0     [003] dN.1  4228.547804: hrtimer_try_to_cancel <-hrtimer_cancel
          <idle>-0     [003] dN.2  4228.547805: hrtimer_force_reprogram <-__remove_hrtimer
          <idle>-0     [003] dN.1  4228.547805: hrtimer_forward <-tick_nohz_idle_exit
          <idle>-0     [003] dN.1  4228.547805: hrtimer_start_range_ns <-hrtimer_start_expires.constprop.11
          <idle>-0     [003] d..1  4228.547858: hrtimer_get_next_event <-get_next_timer_interrupt
          <idle>-0     [003] d..1  4228.547859: hrtimer_start <-__tick_nohz_idle_enter
          <idle>-0     [003] d..2  4228.547860: hrtimer_force_reprogram <-__rem
```

Отметим, что трассировка sys_nanosleep была потеряна.

```
# cat set_ftrace_filter
hrtimer_run_queues
hrtimer_run_pending
hrtimer_init
hrtimer_cancel
hrtimer_try_to_cancel
hrtimer_forward
hrtimer_start
hrtimer_reprogram
hrtimer_force_reprogram
hrtimer_get_next_event
hrtimer_interrupt
hrtimer_nanosleep
hrtimer_wakeup
hrtimer_get_remaining
hrtimer_get_res
hrtimer_init_sleeper
```

Причиной этого стало указание символа > в команде echo. Одиночная угловая скобка (>) задает переписывание файла, а два символа (>>) указывают добавление в конец файла.

Для сброса фильтров с восстановлением записи всех функций можно воспользоваться командой

```
# echo > set_ftrace_filter
# cat set_ftrace_filter
#
```

Далее показано, как можно добавить фильтры, не удаляя имеющихся.

```
# echo sys_nanosleep > set_ftrace_filter
# cat set_ftrace_filter
sys_nanosleep
# echo 'hrtimer_*' >> set_ftrace_filter
# cat set_ftrace_filter
hrtimer_run_queues
hrtimer_run_pending
hrtimer_init
hrtimer_cancel
hrtimer_try_to_cancel
hrtimer_forward
hrtimer_start
hrtimer_reprogram
hrtimer_force_reprogram
hrtimer_get_next_event
hrtimer_interrupt
sys_nanosleep
hrtimer_nanosleep
hrtimer_wakeup
hrtimer_get_remaining
hrtimer_get_res
hrtimer_init_sleeper
```

Фильтр set_ftrace_notrace исключает трассировку указанных в файле функций. Например, команда

```
# echo '*preempt*' '*lock*' > set_ftrace_notrace
```

Приведет к показанному ниже результату.

```
# tracer: function
#
# entries-in-buffer/entries-written: 39608/39608   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
            bash-1994  [000] ....  4342.324896: file_ra_state_init <-do_dentry_open
            bash-1994  [000] ....  4342.324897: open_check_o_direct <-do_last
            bash-1994  [000] ....  4342.324897: ima_file_check <-do_last
            bash-1994  [000] ....  4342.324898: process_measurement <-ima_file_check
            bash-1994  [000] ....  4342.324898: ima_get_action <-process_measurement
            bash-1994  [000] ....  4342.324898: ima_match_policy <-ima_get_action
            bash-1994  [000] ....  4342.324899: do_truncate <-do_last
            bash-1994  [000] ....  4342.324899: should_remove_suid <-do_truncate
            bash-1994  [000] ....  4342.324899: notify_change <-do_truncate
            bash-1994  [000] ....  4342.324900: current_fs_time <-notify_change
            bash-1994  [000] ....  4342.324900: current_kernel_time <-current_fs_time
            bash-1994  [000] ....  4342.324900: timespec_trunc <-current_fs_time
```

Откуда можно видеть, что отслеживания lock и preempt не происходит.

Динамическая трассировка с function_graph

Хотя приведенные выше разъяснения относятся к обоим трассировщикам function и function_graph, последний имеет некоторые особые возможности.

Если вы хотите отслеживать лишь одну функцию и всех ее потомком, достаточно включить имя этой функции с помощью команды echo в файл set_graph_function. Например,

```
echo __do_fault > set_graph_function
```

будет выводить «расширенную» трассировку функции __do_fault().

```
 0)               |  __do_fault() {
 0)               |    filemap_fault() {
 0)               |      find_lock_page() {
 0)   0.804 us    |        find_get_page();
 0)               |        __might_sleep() {
 0)   1.329 us    |        }
 0)   3.904 us    |      }
 0)   4.979 us    |    }
 0)   0.653 us    |    _spin_lock();
 0)   0.578 us    |    page_add_file_rmap();
 0)   0.525 us    |    native_set_pte_at();
 0)   0.585 us    |    _spin_unlock();
 0)               |    unlock_page() {
 0)   0.541 us    |      page_waitqueue();
 0)   0.639 us    |      __wake_up_bit();
 0)   2.786 us    |    }
```

```
  0) + 14.237 us   |  }
  0)               |  __do_fault() {
  0)               |    filemap_fault() {
  0)               |      find_lock_page() {
  0)   0.698 us    |        find_get_page();
  0)               |        __might_sleep() {
  0)   1.412 us    |        }
  0)   3.950 us    |      }
  0)   5.098 us    |    }
  0)   0.631 us    |    _spin_lock();
  0)   0.571 us    |    page_add_file_rmap();
  0)   0.526 us    |    native_set_pte_at();
  0)   0.586 us    |    _spin_unlock();
  0)               |    unlock_page() {
  0)   0.533 us    |      page_waitqueue();
  0)   0.638 us    |      __wake_up_bit();
  0)   2.793 us    |    }
  0) + 14.012 us   |  }
```

Можно таким способом отслеживать несколько функций, например,

```
echo sys_open > set_graph_function
echo sys_close >> set_graph_function
```

Если после этого нужно вернуться к отслеживанию всех функций, можно сбросить фильтр командой

```
echo > set_graph_function
```

ftrace_enabled

Следует отметить, что sysctl ftrace_enable служит основным выключателем для трассировщика функций, который по умолчанию включен (если трассировка функций активизирована в ядре). При отключении трассировки выключается отслеживание всех функций (не только трассировщики функций для ftrace, но и другие - perf, kprobe, трассировка стека, профилирование и т. п.). Следует осторожно относиться к запрету трассировки. Команды управления приведены ниже.

```
sysctl kernel.ftrace_enabled=0
sysctl kernel.ftrace_enabled=1
```

или

```
echo 0 > /proc/sys/kernel/ftrace_enabled
echo 1 > /proc/sys/kernel/ftrace_enabled
```

Команды фильтрации

Интерфейс set_ftrace_filter поддерживает несколько команд, имеющих формат

```
<function>:<command>:<parameter>
```

Ниже перечислены поддерживаемые команды.

mod

Включает фильтрацию функций на уровне модуля, задаваемого параметром. Например, если нужно отслеживать лишь функции write* в модуле ext3, можно воспользоваться командой

```
echo 'write*:mod:ext3' > set_ftrace_filter
```

Эта команда взаимодействует с фильтрами так же, как для фильтрации по именам функций, т. е. для добавления других функций применяется операция записи в конец файла (>>). Для удалении функции применяется восклицательный знак перед именем.

```
echo '!writeback*:mod:ext3' >> set_ftrace_filter
```

Команда позволяет указывать функции и модули с помощью шаблонов. Например, для запрета трассировки всех функций, за исключением указанного модуля, служит команд

```
echo '!*:mod:!ext3' >> set_ftrace_filter
```

Для запрета трассировки всех модулей с сохранением трассировки ядра служит команд

```
echo '!*:mod:*' >> set_ftrace_filter
```

Для фильтрации только функций ядра служат команды вида

```
echo '*write*:mod:!*' >> set_ftrace_filter
```

Для фильтрации набора модулей служат команды вида

```
echo '*write*:mod:*snd*' >> set_ftrace_filter
```

traceon/traceoff

Эти команды включают и выключают трассировку только при обращении к указанным функциям. Параметр определяет число включений и отключений трассировки, при отсутствии параметра число не ограничивается. Например, для запрета трассировки при первых 5 вызовах __schedule_bug можно воспользоваться командой

```
echo '__schedule_bug:traceoff:5' > set_ftrace_filter
```

Для запрета трассировки при вызове __schedule_bug служит команда

```
echo '__schedule_bug:traceoff' > set_ftrace_filter
```

Эти команды являются накопительными независимо от их добавления в set_ftrace_filter. Для удаления команды указывается символ ! Перед ее именем и значение 0 для параметра

```
echo '!__schedule_bug:traceoff:0' > set_ftrace_filter
```

Приведенная выше команда удаляет traceoff для функции __schedule_bug со счетчиком. Для удаления команд бкз счтечиков можно воспользоваться командой вида

```
echo '!__schedule_bug:traceoff' > set_ftrace_filter
```

snapshot

Будет вызывать создание моментального снимка (snapshot) при вызове указанной функции.

```
echo 'native_flush_tlb_others:snapshot' > set_ftrace_filter
```

Для однократного создания снимка служат команды вида

```
echo 'native_flush_tlb_others:snapshot:1' > set_ftrace_filter
```

Для удаления приведенных выше команд можно воспользоваться командами

```
echo '!native_flush_tlb_others:snapshot' > set_ftrace_filter
echo '!native_flush_tlb_others:snapshot:0' > set_ftrace_filter
```

enable_event/disable_event

Эти команды могут включать и выключать трассировку событий. Отметим, что по причине «чувствительности» обратных вызовов трассировки функций при использовании этих команд точки трассировки активируются, но отключаются они в «мягком» (soft) режиме, т. е. точка трассировки будет вызываться, просто не будет отслеживаться ничего. Точка трассировки события остается в таком режиме, пока есть использующие ее команды.

```
echo 'try_to_wake_up:enable_event:sched:sched_switch:2' > set_ftrace_filter
```

Формат команд показан ниже.

```
<function>:enable_event:<system>:<event>[:count]
<function>:disable_event:<system>:<event>[:count]
```

Для удаления связанных с событиями команд служат приведенные ниже команды

```
echo '!try_to_wake_up:enable_event:sched:sched_switch:0' > set_ftrace_filter
```

```
echo '!schedule:disable_event:sched:sched_switch' > set_ftrace_filter
```

dump

При обращении к функции на консоль будет выводиться содержимое кольцевого буфера ftrace. Это полезно при отладке, когда вы хотите быстро увидеть трассировку при обращении к некой функции. Например, ее можно использовать для просмотра буфера перед повторяющимся отказом.

cpudump

При обращении к функции на консоль будет выводиться содержимое кольцевого буфера ftrace для текущего CPU. В отличие от команды dump выводиться будет лишь буфер CPU, на котором выполняется вызвавшая вывод функция.

trace_pipe

Файл trace_pipe выводит те же данные, что и trace, но влияние на трассировку различается. Каждая запись из trace_pipe «потребляется» при считывании, т. е. последовательные операции чтения будут давать новый результат.

```
# echo function > current_tracer
# cat trace_pipe > /tmp/trace.out &
[1] 4153
# echo 1 > tracing_on
# usleep 1
# echo 0 > tracing_on
# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 0/0   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |

#
# cat /tmp/trace.out
           bash-1994  [000] ....  5281.568961: mutex_unlock <-rb_simple_write
           bash-1994  [000] ....  5281.568963: __mutex_unlock_slowpath <-mutex_unlock
           bash-1994  [000] ....  5281.568963: __fsnotify_parent <-fsnotify_modify
           bash-1994  [000] ....  5281.568964: fsnotify <-fsnotify_modify
           bash-1994  [000] ....  5281.568964: __srcu_read_lock <-fsnotify
           bash-1994  [000] ....  5281.568964: add_preempt_count <-__srcu_read_lock
           bash-1994  [000] ...1  5281.568965: sub_preempt_count <-__srcu_read_lock
           bash-1994  [000] ....  5281.568965: __srcu_read_unlock <-fsnotify
           bash-1994  [000] ....  5281.568967: sys_dup2 <-system_call_fastpath
```

Отметим, что чтение файла trace_pipe будет блокироваться, пока в него не добавлено новых данных.

Записи трассировки

При диагностике проблем в ядре избыток или недостаток данных могут создавать проблемы. Фалй buffer_size_kb сшлужит для изменения размера внутренних буферов трассировки. Значение в файле указывает число записей, которые могут быть созданы для каждого CPU. Для определения полного размера это значение следует умножить на число процессоров.

```
# cat buffer_size_kb
1408 (units kilobytes)
```

Или просто прочитать файл buffer_total_size_kb

```
# cat buffer_total_size_kb
5632
```

Для изменения размера буфера просто записывается (echo) нужное число (в сегментах по 1024 байта).

```
# echo 10000 > buffer_size_kb
# cat buffer_size_kb
10000 (units kilobytes)
```

При попытке указать слишком большой размер буферов сработает триггер Out-Of-Memory.

```
# echo 1000000000000 > buffer_size_kb
-bash: echo: write error: Cannot allocate memory
# cat buffer_size_kb
85
```

Буферы per_cpu можно изменять независимо, как показано ниже.

```
# echo 10000 > per_cpu/cpu0/buffer_size_kb
# echo 100 > per_cpu/cpu1/buffer_size_kb
```

При разных размерах буферов per_cpu файл buffer_size_kb на верхнем уровне просто покажет X.

```
# cat buffer_size_kb
X
```

В этом случае можно воспользоваться командой

```
# cat buffer_total_size_kb
12916
```

Запись в buffer_size_kb верхнего уровня будет сбрасывать буферы всех процессоров к одному размеру.

Мгновенные снимки

Опция ядра CONFIG_TRACER_SNAPSHOT включает возможность создания «моментальных снимков» для всех трассировщиков, кроме трассировщиков задержки. Трассировщики задержки, записывающие максимальное значение (такие как irqsoff или wakeup), не могут использовать эту возможность, поскольку в них уже применяется внутренний механизм snapshot mechanism.

Снимок сохраняет текущий буфер трассировки в конкретный момент времени без остановки отслеживания. Ftrace заменяет текущий буфер запасным и трассировка продолжается в новый (ранее резервный) буфер.

Ниже указаны файлы tracefs (tracing), связанные с этой функцией.

snapshot

используется для записи и считывания моментального снимка. Запись (echo) 1 в этот файл ведет к созданию запасного буфера, записи снимка и последующему его считыванию в формате файла (см. раздел Файловая система). Считывание снимка и трассировка выполняются параллельно. Когда запасной буфер создан, запись (echo) 0 в файл освобождает буфер, а запись други (положительных) значений очищает содержимое снимка.

состояние/ввод	0	1	иное
Не выделен	Ничего	Выделение и переключение	Ничего
Выделен	Освобождение	swap	Очистка

Ниже приведен пример использования моментального снимка.

```
# echo 1 > events/sched/enable
# echo 1 > snapshot
# cat snapshot
# tracer: nop
#
# entries-in-buffer/entries-written: 71/71   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          <idle>-0     [005] d...  2440.603828: sched_switch: prev_comm=swapper/5 prev_pid=0 prev_prio=120   prev_state=R ==>
next_comm=snapshot-test-2 next_pid=2242 next_prio=120
           sleep-2242  [005] d...  2440.603846: sched_switch: prev_comm=snapshot-test-2 prev_pid=2242 prev_prio=120   prev_state=R ==>
next_comm=kworker/5:1 next_pid=60 next_prio=120
[...]
        <idle>-0     [002] d...  2440.707230: sched_switch: prev_comm=swapper/2 prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=snapshot-test-2 next_pid=2229 next_prio=120

# cat trace
# tracer: nop
#
# entries-in-buffer/entries-written: 77/77   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          <idle>-0     [007] d...  2440.707395: sched_switch: prev_comm=swapper/7 prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=snapshot-test-2 next_pid=2243 next_prio=120
   snapshot-test-2-2229  [002] d...  2440.707438: sched_switch: prev_comm=snapshot-test-2 prev_pid=2229 prev_prio=120 prev_state=S ==>
next_comm=swapper/2 next_pid=0 next_prio=120
[...]
```

Если попытаться использовать функцию snapshot для одного из трассировщиков задержек, возникнет ошибка.

```
# echo wakeup > current_tracer
# echo 1 > snapshot
bash: echo: write error: Device or resource busy
# cat snapshot
cat: snapshot: Device or resource busy
```

Экземпляры

В каталоге tracing файловой системы tracefs имеется папка instances, где можно создавать новые каталоги с помощью команды mkdir и удалять их командой rmdir. Создаваемые командой mkdir каталоги сразу будут содержать файлы и другие папки.

```
# mkdir instances/foo
# ls instances/foo
buffer_size_kb  buffer_total_size_kb  events  free_buffer  per_cpu
set_event  snapshot  trace  trace_clock  trace_marker  trace_options
trace_pipe  tracing_on
```

Как видно из представленного вывода созданный каталог похож на сам каталог tracing. Фактически он отличается лишь тем, что буфер и события не зависят от основного экземпляра и других созданных экземпляров.

Файлы в новом каталоге работают так же, как одноименные файлы каталога tracing, за исключением того, что для нихе применяется независимый новый буфер. Файлы влияют на этот буфер, но не воздействуют на основной буфер (за исключением trace_options). В настоящее время trace_options одинаково влияет на все экземпляры, включая буфер верхнего уровня, но это может измениться в будущих версиях. Т. е. опции могут стать независимыми для каждого экземпляра.

Отмети, что здесь нет ни одного трассировщика функций, а также файлов current_tracer и available_tracers. Это обусловлено тем, что в настоящее буферы могут включать лишь разрешенные для них события.

```
# mkdir instances/foo
# mkdir instances/bar
# mkdir instances/zoot
# echo 100000 > buffer_size_kb
# echo 1000 > instances/foo/buffer_size_kb
# echo 5000 > instances/bar/per_cpu/cpu1/buffer_size_kb
# echo function > current_trace
# echo 1 > instances/foo/events/sched/sched_wakeup/enable
# echo 1 > instances/foo/events/sched/sched_wakeup_new/enable
# echo 1 > instances/foo/events/sched/sched_switch/enable
# echo 1 > instances/bar/events/irq/enable
# echo 1 > instances/zoot/events/syscalls/enable
# cat trace_pipe
CPU:2 [LOST 11745 EVENTS]
            bash-2044  [002] .... 10594.481032: _raw_spin_lock_irqsave <-get_page_from_freelist
            bash-2044  [002] d... 10594.481032: add_preempt_count <-_raw_spin_lock_irqsave
            bash-2044  [002] d..1 10594.481032: __rmqueue <-get_page_from_freelist
            bash-2044  [002] d..1 10594.481033: _raw_spin_unlock <-get_page_from_freelist
            bash-2044  [002] d..1 10594.481033: sub_preempt_count <-_raw_spin_unlock
            bash-2044  [002] d... 10594.481033: get_pageblock_flags_group <-get_pageblock_migratetype
            bash-2044  [002] d... 10594.481034: __mod_zone_page_state <-get_page_from_freelist
            bash-2044  [002] d... 10594.481034: zone_statistics <-get_page_from_freelist
            bash-2044  [002] d... 10594.481034: __inc_zone_state <-zone_statistics
            bash-2044  [002] d... 10594.481034: __inc_zone_state <-zone_statistics
            bash-2044  [002] .... 10594.481035: arch_dup_task_struct <-copy_process
```

```
  [...]
  # cat instances/foo/trace_pipe
              bash-1998  [000] d..4   136.676759: sched_wakeup: comm=kworker/0:1 pid=59 prio=120 success=1 target_cpu=000
              bash-1998  [000] dN.4   136.676760: sched_wakeup: comm=bash pid=1998 prio=120 success=1 target_cpu=000
            <idle>-0     [003] d.h3   136.676906: sched_wakeup: comm=rcu_preempt pid=9 prio=120 success=1 target_cpu=003
            <idle>-0     [003] d..3   136.676909: sched_switch: prev_comm=swapper/3 prev_pid=0 prev_prio=120 prev_state=R ==>
next_comm=rcu_preempt next_pid=9 next_prio=120
     rcu_preempt-9     [003] d..3   136.676916: sched_switch: prev_comm=rcu_preempt prev_pid=9 prev_prio=120 prev_state=S ==>
next_comm=swapper/3 next_pid=0 next_prio=120
              bash-1998  [000] d..4   136.677014: sched_wakeup: comm=kworker/0:1 pid=59 prio=120 success=1 target_cpu=000
              bash-1998  [000] dN.4   136.677016: sched_wakeup: comm=bash pid=1998 prio=120 success=1 target_cpu=000
              bash-1998  [000] d..3   136.677018: sched_switch: prev_comm=bash prev_pid=1998 prev_prio=120 prev_state=R+ ==>
next_comm=kworker/0:1 next_pid=59 next_prio=120
       kworker/0:1-59    [000] d..4   136.677022: sched_wakeup: comm=sshd pid=1995 prio=120 success=1 target_cpu=001
       kworker/0:1-59    [000] d..3   136.677025: sched_switch: prev_comm=kworker/0:1 prev_pid=59 prev_prio=120 prev_state=S ==>
next_comm=bash next_pid=1998 next_prio=120
  [...]
  # cat instances/bar/trace_pipe
       migration/1-14    [001] d.h3   138.732674: softirq_raise: vec=3 [action=NET_RX]
            <idle>-0     [001] dNh3   138.732725: softirq_raise: vec=3 [action=NET_RX]
              bash-1998  [000] d.h1   138.733101: softirq_raise: vec=1 [action=TIMER]
              bash-1998  [000] d.h1   138.733102: softirq_raise: vec=9 [action=RCU]
              bash-1998  [000] ..s2   138.733105: softirq_entry: vec=1 [action=TIMER]
              bash-1998  [000] ..s2   138.733106: softirq_exit: vec=1 [action=TIMER]
              bash-1998  [000] ..s2   138.733106: softirq_entry: vec=9 [action=RCU]
              bash-1998  [000] ..s2   138.733109: softirq_exit: vec=9 [action=RCU]
              sshd-1995  [001] d.h1   138.733278: irq_handler_entry: irq=21 name=uhci_hcd:usb4
              sshd-1995  [001] d.h1   138.733280: irq_handler_exit: irq=21 ret=unhandled
              sshd-1995  [001] d.h1   138.733281: irq_handler_entry: irq=21 name=eth0
              sshd-1995  [001] d.h1   138.733283: irq_handler_exit: irq=21 ret=handled
  [...]
  # cat instances/zoot/trace
  # tracer: nop
  #
  # entries-in-buffer/entries-written: 18996/18996   #P:4
  #
  #                              _-----=> irqs-off
  #                             / _----=> need-resched
  #                            | / _---=> hardirq/softirq
  #                            || / _--=> preempt-depth
  #                            ||| /     delay
  #           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
  #              | |       |   ||||       |         |
              bash-1998  [000] d...   140.733501: sys_write -> 0x2
              bash-1998  [000] d...   140.733504: sys_dup2(oldfd: a, newfd: 1)
              bash-1998  [000] d...   140.733506: sys_dup2 -> 0x1
              bash-1998  [000] d...   140.733508: sys_fcntl(fd: a, cmd: 1, arg: 0)
              bash-1998  [000] d...   140.733509: sys_fcntl -> 0x1
              bash-1998  [000] d...   140.733510: sys_close(fd: a)
              bash-1998  [000] d...   140.733510: sys_close -> 0x0
              bash-1998  [000] d...   140.733514: sys_rt_sigprocmask(how: 0, nset: 0, oset: 6e2768, sigsetsize: 8)
              bash-1998  [000] d...   140.733515: sys_rt_sigprocmask -> 0x0
              bash-1998  [000] d...   140.733516: sys_rt_sigaction(sig: 2, act: 7fff718846f0, oact: 7fff71884650, sigsetsize: 8)
              bash-1998  [000] d...   140.733516: sys_rt_sigaction -> 0x0
```

Можно видеть, трассировка в буфере верхнего уровня показывает лишь отслеживание функций, а экземпляр foo показывает пробуждения (wakeup) и переключатели задач.

Для удаления экземпляров просто удалите соответствующие каталоги

```
  # rmdir instances/foo
  # rmdir instances/bar
  # rmdir instances/zoot
```

Отметим, что при наличии у процесса открытого в удаляемом экземпляре файла rmdir будетт приводить к отказу с ошибкой EBUSY.

Трассировка стека

Поскольку размер стека ядра фиксирован, важно не исчерпать этот размер вызовами функций. Разработчики ядра должны понимать, что они помещают в стек. Если добавлять туда слишком много, может произойти переполнение стека, которое обычно ведет к аварийному завершению работы системы.

Существуют инструменты, периодически проверяющие использование стека (обычно с прерываниями). Поскольку ftrace обеспечивает трассировку функций, это позволяет проверять размер стека при каждом вызове функции. Это включается через трассировщик стека. Для включения возможности трассировки стека служит конфигурационный параметр ядра CONFIG_STACK_TRACER.

Для включения трассировки стека записывается 1 в файл /proc/sys/kernel/stack_tracer_enabled

```
 # echo 1 > /proc/sys/kernel/stack_tracer_enabled
```

Можно также включить трассировку из командной строки ядра для отслеживания стека в процессе загрузки ядра путем добавления команды stacktrace.

Через несколько минут после активизации трассировки можно посмотреть результаты.

```
  # cat stack_max_size[1]
  2928
  # cat stack_trace
          Depth    Size   Location    (18 entries)
          -----    ----   --------
    0)     2928     224   update_sd_lb_stats+0xbc/0x4ac
    1)     2704     160   find_busiest_group+0x31/0x1f1
    2)     2544     256   load_balance+0xd9/0x662
    3)     2288      80   idle_balance+0xbb/0x130
    4)     2208     128   __schedule+0x26e/0x5b9
    5)     2080      16   schedule+0x64/0x66
    6)     2064     128   schedule_timeout+0x34/0xe0
    7)     1936     112   wait_for_common+0x97/0xf1
    8)     1824      16   wait_for_completion+0x1d/0x1f
    9)     1808     128   flush_work+0xfe/0x119
   10)     1680      16   tty_flush_to_ldisc+0x1e/0x20
   11)     1664      48   input_available_p+0x1d/0x5c
   12)     1616      48   n_tty_poll+0x6d/0x134
   13)     1568      64   tty_poll+0x64/0x7f
```

[1]Эта команда показывает не максимальный размер стека, а его максимальное использование к текущему моменту. Для просмотра максимального размера стека, установленного в ядре, служит команда ulimit -s.

```
  14)     1504     880   do_select+0x31e/0x511
  15)      624     400   core_sys_select+0x177/0x216
  16)      224      96   sys_select+0x91/0xb9
  17)      128     128   system_call_fastpath+0x16/0x1b
```

Отметим, что при использовании опции -mfentry в компиляторе gcc функции будут отслеживаться до того, как они организуют кадр стека. Это означает, что функции конечного уровня (листья - leaf) не будут отслеживаться трассировщиком стека при использовании опции -mfentry. В настоящее время эта опция поддерживается gcc 4.6.0 и выше только на архитектуре x86.

По материалам документации к ядру Linux

Николай Малых

nmalykh@protocols.ru