Modernizing the tasklet API

Источник: https://lwn.net/Articles/830964/

Tasklets offer a deferred-execution method in the Linux kernel; they have been available since the 2.3 development series. They allow interrupt handlers to schedule further work to be executed as soon as possible after the handler itself. The tasklet API has its shortcomings, but it has stayed in place while other deferred-execution methods, including workqueues, have been introduced. Recently, Kees Cook posted a security-inspired patch set (also including work from Romain Perier) to improve the tasklet API. This change is uncontroversial, but it provoked a discussion that might lead to the removal of the tasklet API in the (not so distant) future.

Тасклеты предлагают метод отложенного выполнения в ядре Linux; они доступны начиная с версии 2.3 для разработчиков. Они позволяют обработчикам прерываний планировать дальнейшую работу, которая должна быть выполнена как можно скорее после самого обработчика. У tasklet API есть свои недостатки, но он остался на месте, в то время как были внедрены другие методы отложенного выполнения, включая рабочие очереди. Недавно Кис Кук опубликовал набор исправлений, вдохновленных безопасностью (также включающий работу Ромена Перье), для улучшения tasklet API. Это изменение не вызывает споров, но оно спровоцировало дискуссию, которая может привести к удалению tasklet API в (не столь отдаленном) будущем.

The need for tasklets and other deferred execution mechanisms comes from the way the kernel handles interrupts. An interrupt is (usually) caused by some hardware event; when it happens, the execution of the current task is suspended and the interrupt handler takes the CPU. Before the introduction of threaded interrupts, the interrupt handler had to perform the minimum necessary operations (like accessing the hardware registers to silence the interrupt) and then call an appropriate deferred-work mechanism to take care of just about everything else that needed to be done. Threaded interrupts, yet another import from the realtime preemption work, move the handler to a kernel thread that is scheduled in the usual way; this feature was merged for the 2.6.30 kernel, by which time tasklets were well established.

Потребность в тасклетах и других механизмах отложенного выполнения обусловлена тем, как ядро обрабатывает прерывания. Прерывание (обычно) вызвано каким-либо аппаратным событием; когда это происходит, выполнение текущей задачи приостанавливается, и обработчик прерывания использует центральный процессор. До введения многопоточных прерываний обработчик прерываний должен был выполнять минимально необходимые операции (например, доступ к аппаратным регистрам для отключения прерывания), а затем вызывать соответствующий механизм отложенной работы, чтобы позаботиться практически обо всем остальном, что необходимо было сделать. Потоковые прерывания, еще один импорт из работы с упреждением в реальном времени, перемещают обработчик в поток ядра, который запланирован обычным образом; эта функция была объединена для ядра 2.6.30, к тому времени тасклеты были хорошо установлены.

An interrupt handler will schedule a tasklet when there is some work to be done at a later time. The kernel then runs the tasklet when possible, typically when the interrupt handler finishes, or the task returns to the user space. The tasklet callback runs in atomic context, inside a software interrupt, meaning that it cannot sleep or access user-space data, so not all work can be done in a tasklet handler. Also, the kernel only allows one instance of any given tasklet to be running at any given time; multiple different tasklet callbacks can run in parallel. Those limitations of tasklets are not present in more recent deferred work mechanisms like workqueues. But still, the current kernel contains more than a hundred users of tasklets.

Обработчик прерываний запланирует выполнение тасклета, когда есть какая-то работа, которую необходимо выполнить позже. Затем ядро запускает тасклет, когда это возможно, обычно когда завершается обработка прерывания или задача возвращается в пользовательское пространство. Обратный вызов тасклета выполняется в атомарном контексте, внутри программного прерывания, что означает, что он не может перейти в спящий режим или получить доступ к данным пользовательского пространства, поэтому не вся работа может быть выполнена в обработчике тасклета. Кроме того, ядро позволяет запускать только один экземпляр любого данного тасклета в любой момент времени; несколько различных обратных вызовов тасклета могут выполняться параллельно. Эти ограничения

тасклетов отсутствуют в более поздних механизмах отложенной работы, таких как рабочие очереди. Но, тем не менее, текущее ядро содержит более сотни пользователей тасклетов.

Cook's patch set changes the parameter type for the tasklet's callback. In current kernels, they take an unsigned long value that is specified when the tasklet is initialized. This is different from other kernel mechanisms with callbacks; the preferred way in current kernels is to use a pointer to a type-specific structure. The change Cook proposes goes in that direction by passing the tasklet **context (struct tasklet_struct)** to the callback. The goal behind this work is to avoid a number of problems, including a need to cast from the unsigned int to a different type (without proper type checking) in the callback. The change allows the removal of the (now) redundant data field from the tasklet structure. Finally, this change mitigates the possible buffer overflow attacks that could overwrite the callback pointer and the data field. This is likely one of the primary objectives, as the work was first posted (in 2019) on the kernel-hardening mailing list.

Набор исправлений Кука изменяет тип параметра для обратного вызова тасклета. В текущих ядрах они принимают длинное значение без знака, которое указывается при инициализации тасклета. Это отличается от других механизмов ядра с обратными вызовами; предпочтительным способом в текущих ядрах является использование указателя на структуру, зависящую от типа. Изменение, предлагаемое Куком, направлено в этом направлении путем передачи контекста **тасклета (struct tasklet_struct)** обратному вызову. Цель, стоящая за этой работой, состоит в том, чтобы избежать ряда проблем, включая необходимость преобразования из unsigned int в другой тип (без надлежащей проверки типа) в обратном вызове. Это изменение позволяет удалить (теперь) избыточное поле данных из структуры тасклета. Наконец, это изменение смягчает возможные атаки переполнения буфера, которые могут перезаписать указатель обратного вызова и поле данных. Вероятно, это одна из основных целей, поскольку работа была впервые опубликована (в 2019 году) в списке рассылки по защите ядра.

Plotting the removal of tasklets

Построение графика удаления тасклетов

The patch set caused no controversies, but the discussion changed direction following this comment from Peter Zijlstra, who said: "I would _MUCH_ rather see tasklets go the way of the dodo [...] Can't we stage an extinction event here instead?" In a response, Sebastian Andrzej Siewior suggested that tasklets could be replaced with threaded interrupts, as they also run in atomic context. Dmitry Torokhov suggested immediately expiring timers instead. Cook replied that the change could not be done mechanically and gave some examples of more complicated usage of tasklets. One such case is the AMD ccp crypto driver, which combines tasklets with DMA engines, while another is the Intel i915 GPU driver, which schedules GPU tasks with tasklets.

Набор исправлений не вызвал разногласий, но обсуждение изменило направление после этого комментария Питера Зейлстры, который сказал: "Я бы предпочел, чтобы тасклеты пошли по пути додо [...] Разве мы не можем вместо этого провести здесь мероприятие по вымиранию?" В ответ Себастьян Анджей Сивиор предположил, что тасклеты можно заменить потоковыми прерываниями, поскольку они также выполняются в атомарном контексте. Дмитрий Торохов предложил вместо этого немедленно отключить таймеры. Кук ответил, что это изменение невозможно выполнить механически, и привел несколько примеров более сложного использования тасклетов. Одним из таких примеров является драйвер AMD ccp crypto, который объединяет тасклеты с движками DMA, в то время как другим является драйвер графического процессора Intel i915, который планирует задачи графического процессора с помощью тасклетов.

In the following messages, Thomas Gleixner "grudgingly" acked the patch set, but also spoke in favor of removing tasklets: "I'd rather see tasklets vanish from the planet completely, but that's going to be a daring feat." The developers agreed that removing tasklets would be a logical next step, but that this is a bigger task than improving their API. The Kernel Self-Protection Project has added a dedicated task for this objective.

В следующих сообщениях Томас Глейкснер "неохотно" одобрил набор исправлений, но также высказался за удаление тасклетов: "Я бы предпочел, чтобы тасклеты полностью исчезли с лица

планеты, но это будет смелый подвиг". Разработчики согласились с тем, что удаление тасклетов было бы логичным следующим шагом, но это более сложная задача, чем улучшение их API. Проект самозащиты ядра добавил специальную задачу для достижения этой цели.

The removal of the tasklet API has been discussed before; LWN covered it in 2007. At that time, the main argument for the removal of tasklets was to limit latencies (since tasklets run in software interrupt mode, they can block even the highest-priority tasks). The argument against removing tasklets was a possible performance loss for drivers that need to react quickly to events. At that time, threaded interrupts were not yet included in the mainline.

In current kernels, tasklets can be replaced by workqueues, timers, or threaded interrupts. If threaded interrupts are used, the work may just be executed in the interrupt handler itself. Those newer mechanisms do not have the disadvantages of tasklets and should satisfy the same needs, so developers do not see a reason to keep tasklets. It seems that any migration away from tasklets will be done one driver (or subsystem) at a time. For example, Takashi Iwai already reported having the conversion ready for sound drivers.

Удаление tasklet API обсуждалось ранее; LWN освещал это в 2007 году. В то время основным аргументом в пользу удаления тасклетов было ограничение задержек (поскольку тасклеты выполняются в режиме программного прерывания, они могут блокировать даже задачи с самым высоким приоритетом). Аргументом против удаления тасклетов была возможная потеря производительности для драйверов, которым необходимо быстро реагировать на события. В то время многопоточные прерывания еще не были включены в основную линию.

В современных ядрах тасклеты могут быть заменены рабочими очередями, таймерами или потоковыми прерываниями. Если используются потоковые прерывания, работа может быть просто выполнена в самом обработчике прерываний. Эти новые механизмы не обладают недостатками тасклетов и должны удовлетворять тем же потребностям, поэтому разработчики не видят причин сохранять тасклеты. Похоже, что любой переход от тасклетов будет выполняться по одному драйверу (или подсистеме) за раз. Например, Такаши Иваи уже сообщил о готовности преобразования для звуковых драйверов.

Current API changes

While the removal of tasklets remains a longer-term goal, the developers are proceeding with the API changes. The modifications in the tasklet API performed by Cook's patch set are minimal and consist of creating a new initialization macro and adding one initialization function. In current kernels, tasklets are declared with:

Хотя удаление тасклетов остается долгосрочной целью, разработчики продолжают вносить изменения в API. Модификации в tasklet API, выполняемые Cook's patch set, минимальны и состоят из создания нового макроса инициализации и добавления одной функции инициализации. В текущих ядрах тасклеты объявляются с помощью:

```
    #define DECLARE_TASKLET(name, func, data) \
struct tasklet_struct name = {NULL,0,ATOMIC_INIT(0),func, data}
```

To allow compatibility with existing users, all calls to the "old" DECLARE_TASKLET() were changed to DECLARE_TASKLET_OLD with the following definition:

```
#define DECLARE_TASKLET_OLD(name, _func)        \
     struct tasklet_struct name = {             \
     .count = ATOMIC_INIT(0),                   \
     .func = _func,                             \
}
```

The same modifications were done to the DECLARE_TASKLET_DISABLED() macro. The conversion to DECLARE_TASKLET_OLD() turned out to be mechanical, since all those users provided zero as the data parameter.

A following patch included a new version of the declaration macro that does not contain that data parameter:

```
#define DECLARE_TASKLET(name, _callback)        \
     struct tasklet_struct name = {             \
     .count = ATOMIC_INIT(0),                   \
     .callback = _callback,                     \
     .use_callback = true,                      \
}
```

In the new API, the callback function is stored in the callback() field rather than func(); the callback itself simply takes a pointer to the tasklet_struct structure as its one argument:

```
void (*callback)(struct tasklet_struct *t);
```

That structure will normally be embedded within a larger, user-specific structure, the pointer to which can be obtained with the container_of() macro in the usual way. The patch set also adds a function to initialize a tasklet at run time, with the following prototype:

Эта структура обычно будет встроена в более крупную, специфичную для пользователя структуру, указатель на которую можно получить с помощью макроса container_of() обычным способом. Набор исправлений также добавляет функцию для инициализации тасклета во время выполнения со следующим прототипом:

```
void tasklet_setup(struct tasklet_struct *t,
     void (*callback)(struct tasklet_struct *));
```

The tasklet subsystem will invoke the callback in either the new or the old mode, depending on how the tasklet was initialized; beyond that, the behavior of tasklets is unchanged.

Подсистема тасклетов вызовет обратный вызов либо в новом, либо в старом режиме, в зависимости от того, как был инициализирован тасклет; кроме того, поведение тасклетов остается неизменным.

Where to from here

Куда направляемся отсюда

The team working on the change submitted a number of patches to convert all tasklet initializations in the kernel to the new tasklet_setup() function. Another task remains to remove the tasklets from all those users. The work in some subsystems has already started. Developers are welcome to help with the conversion of all subsystems to the new API and, eventually, removing all tasklet users from the kernel. There is certainly plenty of will on the part of the kernel developers to do so, but this is likely going to take a few kernel development cycles.

Команда, работающая над изменением, представила ряд исправлений для преобразования всех инициализаций тасклетов в ядре в новую функцию tasklet_setup(). Остается еще одна задача - удалить тасклеты у всех этих пользователей. Работа в некоторых подсистемах уже началась. Разработчики могут помочь с преобразованием всех подсистем в новый API и, в конечном счете, удалением всех пользователей тасклетов из ядра. Безусловно, у разработчиков ядра есть большая воля сделать это, но это, вероятно, займет несколько циклов разработки ядра.