Анализ типовых ошибок в драйверах операционной системы Linux[1]

В.С. Мутилин, Е.М. Новиков, А.В. Хорошилов
mutilin@ispras.ru, joker@ispras.ru, khoroshilov@ispras.ru

Аннотация. Быстрый темп развития ядра и драйверов операционной системы Linux, разрабатываемых большим распределенным сообществом программистов, привел к тому, что на сегодняшний день не существует единой базы правил, которые полностью описывают корректное взаимодействие драйверов и ядра. Это является препятствием, с одной стороны, для тех программистов, которые не обладают экспертными знаниями во всех особенностях данного взаимодействия; с другой стороны, для разработки и применения инструментов, которые могли бы находить соответствующие типовые ошибки автоматизированным образом. В данной статье предлагается методика выявления и классификации типовых ошибок и соответствующих им правил на основе изменений, вносимых в драйверы операционной системы Linux. В статье приводятся результаты применения данной методики, обсуждаются полученная классификация и распределение типовых ошибок по классам.

Ключевые слова: операционная система; ядро; драйвер; правило взаимодействия; классификация ошибок.

1. Введение

Ядро операционной системы (ОС) Linux является одним из самых динамично развивающихся и востребованных проектов в мире. Разработку ядра начал Линус Торвальдс в 1991 году. В настоящее время в подготовке каждого нового релиза ядра ОС Linux участвуют более 1000 человек, распределенных по всему миру [1]. Релизы выпускаются в среднем раз в 2-3 месяца. Начиная с ядра версии 2.6.24, выпущенного в начале 2008 года, каждый релиз включает порядка 9-12 тысяч изменений, что соответствует в среднем 5.4 изменениям в час. Размер исходного кода последней на сегодняшний день стабильной версии ядра 3.3 составляет более 15 млн. строк кода [2].

Официальное, так называемое оригинальное (от англ. mainline), ядро ОС Linux выпускает Линус Торвальдс [3]. Процесс разработки оригинального ядра устроен следующим образом. После того, как выпущена очередная версия

[1] Работа поддержана ФЦП "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы" (контракт N 07.514.11.4104)

ядра, например, 2.6.36, начинается период активной разработки («merge window»). В ходе этого периода делается множество изменений, в основном, связанных со слиянием новой функциональности и исправлением ошибок, которые уже были обкатаны в ветках разработки соответствующих подсистем и в тестовом ядре (Рис. 1) [4].

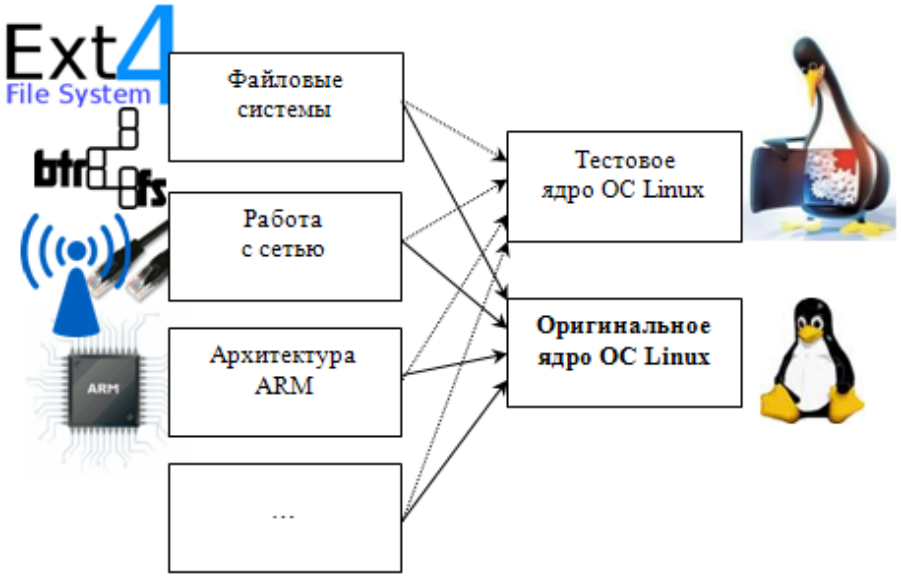

Рис. 1. Процесс формирования оригинального ядра ОС Linux

Через две-три недели период активной разработки завершается выпуском релиз-кандитата («release candidate») 2.6.37-rc1. По мере отладки выпускается серия релиз-кандитатов 2.6.37-rc2, 2.6.37-rc3 и т.д. После того, как все существенные проблемы исправлены, выпускается новая версия оригинального ядра 2.6.37 и начинается период активной разработки следующей версии 2.6.38. Одновременно с этим формируется специальная ветка для поддержки ядра версии 2.6.37, которая получает статус стабильной. В нее, как правило, попадают критические исправления из оригинальной ветки ядра. На основе этих изменений формируются стабильные версии 2.6.37.1, 2.6.37.2 и т.д. Помимо исправлений в стабильные версии попадает небольшое количество изменений, связанных с активацией поддержки новых устройств. Последнее означает, если в ядре уже реализована поддержка определенного семейства устройств, то при выпуске нового устройства из этого семейства в стабильную ветку ядра может попасть исправление, связывающее идентификатор устройства с драйвером семейства. По умолчанию, стабильная версия поддерживается до выпуска следующей версии оригинального ядра, но для отдельных версий продолжительность поддержки

может быть значительно продлена. Наглядно нумерация версий оригинального ядра ОС Linux представлена на **Ошибка! Источник ссылки не найден.**.

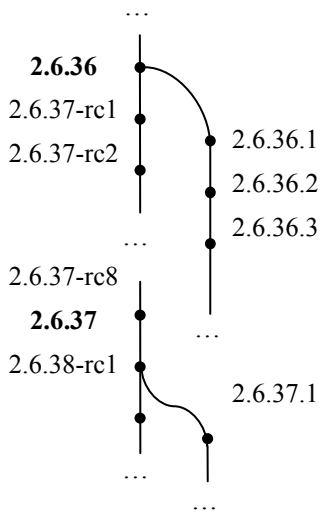

Рис. 2. Нумерация версий оригинального ядра ОС Linux

С мая 2011 года политика нумерации была изменена. Вместо ядра версии 2.6.40 было выпущено ядро версии 3.0. С этого момента для нумерации версий ядра вместо четырех цифр стали использовать три, причем вторая цифра играет роль бывшей третьей, третья – бывшей четвертой.

Существует большое количество других веток разработки ядра, которые основываются на оригинальном ядре ОС Linux (**Ошибка! Источник ссылки не найден.**). Как правило, разработчики различных дистрибутивов Linux поддерживают свои версии ядра. Об этом говорят, например, разработчики дистрибутивов Red Hat Enterprise Linux [5], openSUSE [6] и Debian [7]. Данные версии ядра отличаются от оригинальной тем, что в них поддерживается некоторая дополнительная функциональность и/или содержатся исправления ошибок. Есть ветки ядра, в которых основы операционной системы реализуются принципиально иначе по сравнению с оригинальной версией [8], [9]. С течением времени изменения, интересные широкому кругу лиц, из различных веток разработки попадают в оригинальное ядро [2], [4].

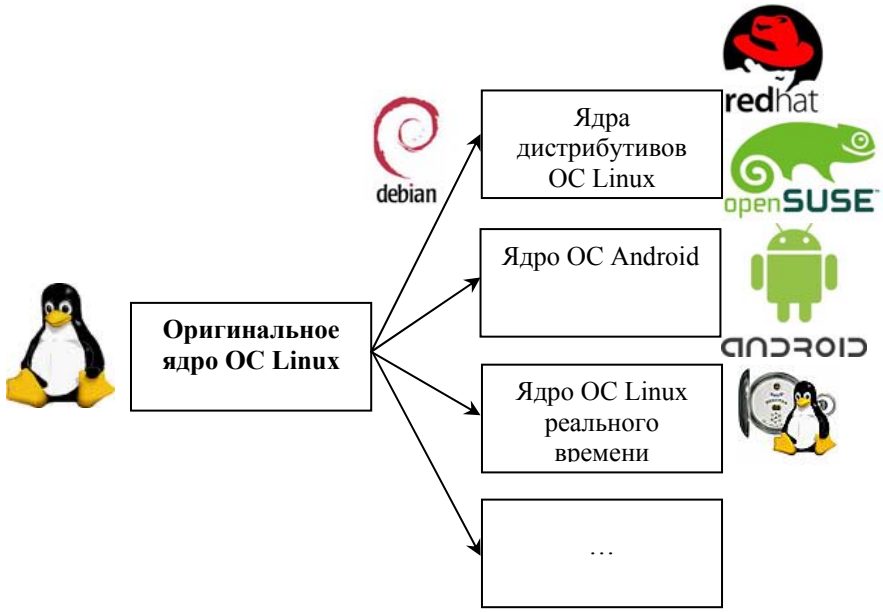

Рис. 3. Ядра, основывающиеся на оригинальном ядре ОС Linux

Схема устройства ядра ОС Linux представлена на Рис. 4. Ядро состоит из основной части («сердцевины») и драйверов устройств. Драйверы устройств участвуют во взаимодействиях:

- с сердцевиной ядра, обрабатывая события и запросы, связанные с соответствующим устройством, и вызывая «библиотечные» функции из сердцевины;
- с аппаратурой, которую они представляют в ядре;
- с пользовательскими приложениями посредством специальных файлов (procfs, sysfs или debugfs).

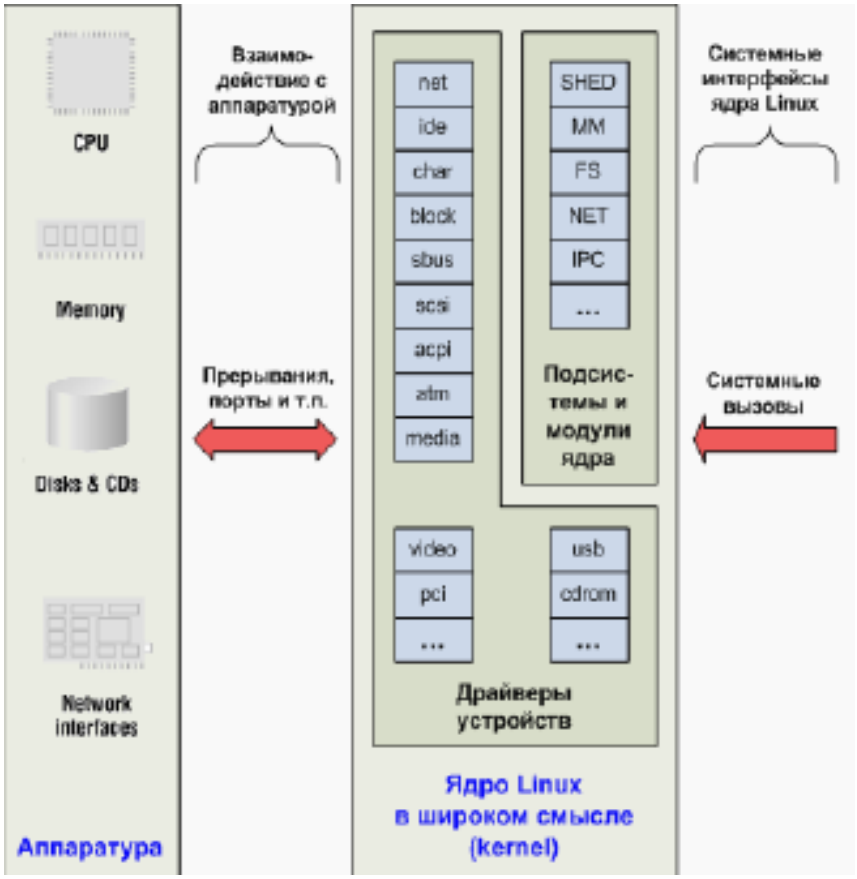

Рис. 4. Схема устройства ядра операционной системы Linux

Драйверы занимают наибольшую часть (до 70%) ядра ОС Linux. В исходном коде драйверов содержится достаточно большое количество различных ошибок, приводящих к некорректной работе всей ОС, зависаниям и падениям. Результаты исследований, которые были проделаны в работах [10] и [11] в начале 2000-х годов для ядер версий от 1.0 до 2.4.1, показали, что драйверы содержат до 85% всех ошибок, которые встречаются во всем ядре ОС Linux. Подобное исследование для ядра ОС Microsoft Windows XP в 2006 году также показало, что наибольшее количество ошибок в ядре данной ОС происходит в драйверах устройств [12]. Более поздние исследования, проведенные в 2011 году для ядер ОС Linux версий от 2.6.0 до 2.6.33, продемонстрировали, что хотя количество ошибок в драйверах стало меньше, чем в компонентах ядра, отвечающих за поддержку различных архитектур и файловых систем, их по-прежнему достаточно много [13].

Задача обеспечения надежности драйверов очень актуальна, так как драйверы работают с тем же уровнем привилегий, что и остальное ядро ОС Linux. Из-за этого, например, недоверенные пользовательские приложения могут использовать уязвимости в драйверах, чтобы исполнять произвольный код с привилегиями ядра и получать доступ к структурам ядра, что отмечено в работах [14] и [15]. В данных работах предлагаются автоматизированные решения, нацеленные на решение данной проблемы. Однако, с одной стороны, они не могут гарантировать полную защиту, требуют дополнительных настроек и т.п. С другой стороны, они могут достаточно сильно снизить производительность работы, что зачастую бывает недопустимо.

Ядро и большая часть драйверов ОС Linux, как входящих, так и не входящих в ядро, являются свободным программным обеспечением. Это, в частности, предоставляет широкие возможности по анализу и тестированию, в том числе на уровне исходного кода. В тестировании ядра и драйверов принимают участие огромное количество пользователей ОС Linux. То, что много людей имеют возможность просматривать исходный код ядра и драйверов ОС Linux с различных точек зрения создает эффект «многих глаз» [16]. Это позволяет обнаруживать и исправлять достаточно большое количество ошибок за короткое время. Тем не менее, данный подход не может гарантировать обнаружение всех ошибок в силу того, что только небольшое количество людей обладают достаточным уровнем знания во всех особенностях сложных взаимодействий между компонентами ядра, а также потому что на исходный код драйверов смотрят меньше, чем на исходный код сердцевины ядра [17].

Ошибки в драйверах можно условно разделить на типовые и нетиповые. К нетиповым относятся ошибки, которые связаны с некорректным взаимодействием с соответствующими устройствами и нарушениями контрактов. Нетиповые ошибки выделяются среди других ошибок тем, что они затрагивают характерные только для некоторого драйвера константы, ограничения, вычисления и т.д. Особенность типовых ошибок заключается в том, что для них можно выделить соответствующие правила, общие для всех драйверов либо для группы драйверов.

Среди типовых ошибок можно выделить три класса. В первый класс входят общие ошибки в драйверах, как в программах на языке программирования Си (ядро и драйверы ОС Linux разрабатываются на языке программирования Си). Например, разыменовывание нулевого указателя, превышение максимально возможных значений переменных с целым типом и т.п. Ко второму классу относятся специфичные ошибки, связанные с неправильным использованием интерфейса сердцевины ядра. К числу таких ошибок относятся, например, нарушения правил инициализации переменных, имеющих специфичные типы. Третий класс типовых ошибок включает в себя состояния гонок и взаимные блокировки, которые возникают при параллельном выполнении вследствие неиспользования или неправильного использования механизмов синхронизации.

Для типовых общих ошибок существуют классификации [18]. Правила, вследствие нарушения которых они происходят, практически не меняются со временем (вообще говоря, язык программирования Си и его расширения продолжают развиваться, что приводит к незначительным изменениям в общих правилах). Ошибки, связанные с параллельным выполнением, также достаточно стандартные и мало меняются с течением времени.

Типовые специфичные ошибки не являются стандартными. Правила, соответствующие им меняются, удаляются и появляются достаточно часто ввиду того, что разработка ядра ОС Linux, как уже отмечалось, ведется большими темпами, чтобы отвечать современным направлениям развития и поддерживать новые особенности архитектур и устройств, в том числе, меняется интерфейс сердцевины ядра. По заявлению Грега Кроа-Хартмана, одного из ведущих разработчиков ядра ОС Linux, предполагается, что интерфейс основной части ядра является нестабильным [19].

Целью данной работы является выявление и классификация типовых ошибок и соответствующих им правил в драйверах операционной системы Linux. В разделе 2 данной статьи предложена методика выявления и классификации типовых ошибок. Раздел 3 описывает результаты практического применения данной методики для драйверов ОС Linux. Аналогичные подходы рассмотрены в разделе 4. В заключении подводятся итоги проведенной работы и приводятся возможные направления дальнейшего развития.

2. Методика выявления и классификации типовых ошибок в драйверах ОС Linux

2.1. Выбор источника типовых ошибок

Выявлять правила, соответствующие типовым ошибкам в драйверах ОС Linux, в особенности, ошибкам некорректного использования интерфейса сердцевины ядра, можно на основе различных источников. Достаточно много информации можно найти в документации, в том числе, входящей в состав ядра [20] и в исходный код ядра и драйверов, а также в литературе по разработке драйверов и ядра ОС Linux (см., например, [21], [22]). Однако, в этих источниках документированы далеко не все особенности различных подсистем ядра и типов драйверов. Поскольку ядро развивается очень стремительно, данные источники не всегда поддерживаются в актуальном состоянии.

В качестве еще одного источника для выявления типовых ошибок можно рассмотреть список рассылки ядра Linux (от англ. Linux Kernel Mailing List, LKML) [23]. В данном списке рассылки обсуждаются различные актуальные вопросы по разработке ядра, в том числе, касательно исправления ошибок. На основании этих обсуждений можно выявить множество типовых общих и специфичных ошибок. Анализ сообщений в LKML достаточно трудоемок,

поскольку сообщений очень много и они содержат большое количество информации, причем не только технической. Кроме того, много ошибок обсуждается в других списках рассылок, на форумах, системах отслеживания ошибок и т.д.

Типовые ошибки и правила можно выявлять на основе анализа изменений, вносимых в драйверы ОС Linux. Данный источник является достаточно актуальным, поскольку раньше или позже среди изменений появляются все важные исправления ошибок, которые обсуждаются в различных списках рассылок, на конференциях, форумах и т.д. Изменения в драйверах ОС Linux проходят достаточно тщательную предварительную процедуру согласования и проверки, в том числе, у экспертов в соответствующих подсистемах ядра [4].

Каждое изменение содержит в себе достаточно подробную информацию, включающую, авторов и время, краткое название изменения, подробное описание (возможно, со ссылками на соответствующие обсуждения), изменение исходного кода драйверов и/или ядра ОС Linux и различную вспомогательную информацию (см. примеры далее в данном разделе). В рамках данной статьи изменения, вносимые в драйверы ОС Linux, были выбраны в качестве источника для анализа типовых ошибок в драйверах.

2.2. Методика выявления и классификации типовых ошибок

Первый шаг методики анализа изменения в драйвере ОС Linux заключается в том, чтобы определить, является ли оно исправлением ошибки или оно каким-либо образом расширяет функциональность драйвера (например, добавляет поддержку новых устройств). Для изменений, которые являются исправлением ошибок, необходимо определить типовые или нетиповые ошибки они исправляют. Для этого необходимо проанализировать описание исправления и изменение исходного кода. В том случае, если ошибка связана с неправильным использованием специфичных для драйвера или группы драйверов констант, условий и вычислений, ее необходимо отнести к нетиповым. Иначе – к типовым.

Основным шагом предлагаемой методики анализа изменений в драйверах ОС Linux является классификация типовых ошибок. Данный шаг требует осмысления причины, которая привела к типовой ошибке. Дело в том, что с первого взгляда может показаться, что ошибка проявилась вследствие нарушения некого общего правила. На самом деле, причина ошибки может крыться в особенностях параллельного выполнения либо в некорректном использовании интерфейса сердцевины ядра. Кроме того, не всегда легко определить, какую именно общую или специфичную ошибку исправляет рассматриваемое изменение. На данном шаге помимо анализа описания и изменения в исходном коде предполагается более тщательный анализ взаимодействий различных подсистем ядра и драйвера, для чего можно использовать, например, [24].

Важно отметить, что список классов не известен заранее и может быть определен только непосредственно по ходу самого анализа. Предполагается, что, в первую очередь, классы позволят различить общие и специфичные ошибки в драйверах, а также ошибки, связанные с параллельным выполнением. Общие ошибки и ошибки, связанные с параллельным выполнением, должны быть разделены посредством подклассов на некоторые хорошо устоявшиеся и общепринятые группы, например, наподобие [18]. Для специфичных ошибок подклассы должны показывать их характерные особенности, а также указывать на затрагиваемые подсистемы сердцевины ядра ОС Linux. Подобное разбиение позволит более точно классифицировать типовые ошибки и построить распределение типовых ошибок по различным классам и подклассам.

Заключительный шаг методики по анализу изменения в драйвере ОС Linux – это описание характерных особенностей изменения посредством вспомогательного комментария. В дальнейшем комментарии помогут быстрее понимать суть проанализированных изменений и вносить некоторые поправки в классификацию при необходимости.

2.3. Примеры применения предложенной методики

Рассмотрим применение предложенной методики на примерах. На основании описания и изменения в исходном коде для изменения в драйвере, представленного на **Ошибка! Источник ссылки не найден.** можно сделать вывод, что данное изменение в драйвере не является исправлением ошибки, а добавляет поддержку нового USB устройства. В качестве комментария к данному изменению можно написать «*Add WAGO 750-923 Service Cable device ID*».

```
commit 5fc8fe8e2ea30805bee5a13420817d6ad34ea9ce
Author: Anders Larsen <al@alarsen.net>
Date:   Wed Oct 6 23:46:25 2010 +0200

    USB: cp210x: Add WAGO 750-923 Service Cable device ID

    commit 93ad03d60b5b18897030038234aa2ebae8234748 upstream.

    The WAGO 750-923 USB Service Cable is used for configuration and
firmware
    updates of several industrial automation products from WAGO
Kontakttechnik GmbH.

    (skipped)

    Signed-off-by: Anders Larsen <al@alarsen.net>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/drivers/usb/serial/cp210x.c b/drivers/usb/serial/cp210x.c
index 3ad53bd..9927bca 100644
```



```
--- a/drivers/usb/serial/cp210x.c
+++ b/drivers/usb/serial/cp210x.c
@@ -132,6 +132,7 @@ static const struct usb_device_id id_table[] = {
        {  USB_DEVICE(0x17F4,  0xAAAA)  },  /*  Wavesense  Jazz  blood
glucose meter */
        {  USB_DEVICE(0x1843,  0x0200)  },  /*  Vaisala  USB  Instrument
Cable */
        { USB_DEVICE(0x18EF, 0xE00F) }, /* ELV USB-I2C-Interface */
+       { USB_DEVICE(0x1BE3, 0x07A6) }, /* WAGO 750-923 USB Service
Cable */
        { USB_DEVICE(0x413C, 0x9500) }, /* DW700 GPS USB interface */
        { } /* Terminating Entry */
 };
```

Рис. 5. Описание изменения 5fc8fe8, добавляющего поддержку нового устройства

Изменение в драйвере на Рис. демонстрирует исправление нетиповой ошибки, вызванной нарушением контракта драйвера трекпада. Изменение исходного кода задействует специфичные для данного драйвера константы и ограничения. Комментарий может быть следующий: «*Ignore the relative axes from the Magic Trackpad*».

```
commit 8e6b41c76c5b8a27b2abd7b9f6ed0877987fd11b
Author: Chase Douglas <chase.douglas@canonical.com>
Date:   Tue Jan 11 19:37:50 2011 +0100

    HID: magicmouse: Don't report REL_{X, Y} for Magic Trackpad

    [ Linus' tree commit 6a66bbd693c12f71697c61207aa18bc5a12da0ab ]

    With the recent switch to having the hid layer handle standard
axis
    initialization, the Magic Trackpad now reports relative axes. This
would
    be fine in the normal mode, but the driver puts the device in
multitouch
    mode where no relative events are generated. Also, userspace
software
    depends on accurate axis information for device type detection.
Thus,
    ignoring the relative axes from the Magic Trackpad is best.

    Signed-off-by: Chase Douglas <chase.douglas@canonical.com>
    Signed-off-by: Jiri Kosina <jkosina@suse.cz>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff    --git    a/drivers/hid/hid-magicmouse.c    b/drivers/hid/hid-
magicmouse.c
index e6dc151..ed732b7 100644
--- a/drivers/hid/hid-magicmouse.c
+++ b/drivers/hid/hid-magicmouse.c
@@  -433,6  +433,11  @@  static  int  magicmouse_input_mapping(struct
hid_device *hdev,
        if (!msc->input)
```

```
                msc->input = hi->input;

+       /* Magic Trackpad does not give relative data after switching
to MT */
+       if (hi->input->id.product == USB_DEVICE_ID_APPLE_MAGICTRACKPAD
&&
+           field->flags & HID_MAIN_ITEM_RELATIVE)
+               return -1;
+
        return 0;
 }
```

Рис.6. Описание изменения 8e6b41c, исправляющего нетиповую ошибку

На **Ошибка! Источник ссылки не найден.** представлено исправление типовой общей ошибки, переполнения целых чисел. В качестве короткого комментария к данному изменению можно взять часть его описания, например: «*An integer overflow occurs in the calculation of RHlinear when the relative humidity is greater than around 30%*».

```
commit 4e3837bb22675cef56b921230f7604fe1a451a5f
Author: Vivien Didelot <vivien.didelot@savoirfairelinux.com>
Date:   Mon Mar 21 17:59:35 2011 +0100

    hwmon: (sht15) Fix integer overflow in humidity calculation

    commit ccd32e735de7a941906e093f8dca924bb05c5794 upstream.

    An integer overflow occurs in the calculation of RHlinear when the
    relative humidity is greater than around 30%. The consequence is a
subtle
    (but noticeable) error in the resulting humidity measurement.

    Signed-off-by:                    Vivien                    Didelot
<vivien.didelot@savoirfairelinux.com>
    Signed-off-by: Jean Delvare <khali@linux-fr.org>
    Cc: Jonathan Cameron <jic23@cam.ac.uk>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/drivers/hwmon/sht15.c b/drivers/hwmon/sht15.c
index a610e78..38a41d2 100644
--- a/drivers/hwmon/sht15.c
+++ b/drivers/hwmon/sht15.c
@@ -333,11 +333,11 @@ static inline int sht15_calc_humid(struct
sht15_data *data)

        const int c1 = -4;
        const int c2 = 40500; /* x 10 ^ -6 */
-       const int c3 = -2800; /* x10 ^ -9 */
+       const int c3 = -28; /* x 10 ^ -7 */

        RHlinear = c1*1000
```



```
                + c2 * data->val_humid/1000
-               + (data->val_humid * data->val_humid * c3)/1000000;
+               + (data->val_humid * data->val_humid * c3) / 10000;
        return (temp - 25000) * (10000 + 80 * data->val_humid)
                / 1000000 + RHlinear;
}
```

Рис.7. Описание изменения 4e3837b, исправляющего типовую общую ошибку (переполнение целых чисел)

На Рис. представлено изменение в драйвере, которое является исправлением типовой специфичной ошибки. Данная ошибка вызвана неправильным использованием интерфейса сердцевины ядра, а именно, в контексте блокировки вызывается функция выделения памяти, которая может заснуть [25]. Основываясь на описании изменения, возможно написать следующий комментарий: «*AppArmor may do a GFP_KERNEL memory allocation while holding lock*».

```
commit 83a9a8034ee98ac21804c376ec90af8e4997790e
Author: John Johansen <john.johansen@canonical.com>
Date:   Wed Jun 8 15:07:47 2011 -0700

    AppArmor: Fix sleep in invalid context from task_setrlimit

    commit 1780f2d3839a0d3eb85ee014a708f9e2c8f8ba0e upstream.

    Affected kernels 2.6.36 - 3.0

    AppArmor may do a GFP_KERNEL memory allocation with task_lock(tsk-
>group_leader);
    held when called from security_task_setrlimit.   This will only
occur when the
    task's current policy has been replaced, and the task's creds have
not been
    updated before entering the LSM security_task_setrlimit() hook.

    BUG:   sleeping   function   called   from   invalid   context   at
mm/slub.c:847
     in_atomic(): 1, irqs_disabled(): 0, pid: 1583, name: cupsd

    (skipped)

    Signed-off-by: John Johansen <john.johansen@canonical.com>
    Reported-by: Miles Lane <miles.lane@gmail.com>
    Signed-off-by: James Morris <jmorris@namei.org>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/security/apparmor/lsm.c b/security/apparmor/lsm.c
index ec1bcec..3d2fd14 100644
--- a/security/apparmor/lsm.c
+++ b/security/apparmor/lsm.c
@@ -612,7 +612,7 @@ static int apparmor_setprocattr(struct task_struct
```

```
*task, char *name,
 static int apparmor_task_setrlimit(struct task_struct *task,
                unsigned int resource, struct rlimit *new_rlim)
 {
-       struct aa_profile *profile = aa_current_profile();
+       struct aa_profile *profile = __aa_current_profile();
        int error = 0;

        if (!unconfined(profile))
```

Рис.8. Описание изменения 83a9a80, исправляющего типовую специфичную ошибку (вызов функции, которая может заснуть, в контексте блокировки)

Рис. демонстрирует типовую ошибку с общим проявлением (использование памяти после ее освобождения) при том, что ее причина заключается в некорректном использовании спин блокировки при параллельном выполнении. Комментарий к данному изменению может быть следующий: «*Calls to ath9k_hw_startpcureceive should be performed under lock*».

```
commit 43d7d3dec4052bb98790a26f67abbb01b5e1b9f3
Author: Luis R. Rodriguez <lrodriguez@atheros.com>
Date:   Wed Oct 20 16:07:04 2010 -0700

    ath9k: add locking for starting the PCU on RX

    commit 7583c550c3e635dcc61ab127c36ecefd59fb8dc8 upstream.

    There was some locking for starting some parts of
    RX but not for starting the PCU. Include this otherwise
    we can content against stopping the PCU.

    This can potentially lead to races against different
    buffers on the PCU which can lead to to the DMA RX
    engine writing to buffers which are already freed.

    (skipped)

    Cc: Ben Greear <greearb@candelatech.com>
    Cc: Kyungwan Nam <kyungwan.nam@atheros.com>
    Signed-off-by: Luis R. Rodriguez <lrodriguez@atheros.com>
    Tested-by: Ben Greear <greearb@candelatech.com>
    Signed-off-by: John W. Linville <linville@tuxdriver.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

    index e581b1f..b3c9baf 100644

diff           --git           a/drivers/net/wireless/ath/ath9k/recv.c
b/drivers/net/wireless/ath/ath9k/recv.c
index 8b1b459..2a2b3f7 100644
--- a/drivers/net/wireless/ath/ath9k/recv.c
+++ b/drivers/net/wireless/ath/ath9k/recv.c
@@ -288,11 +288,11 @@ static void ath_edma_start_recv(struct ath_softc
```



```
*sc)
        ath_rx_addbuffer_edma(sc, ATH9K_RX_QUEUE_LP,
                              sc-
>rx.rx_edma[ATH9K_RX_QUEUE_LP].rx_fifo_hwsize);

-       spin_unlock_bh(&sc->rx.rxbuflock);
-
        ath_opmode_init(sc);

        ath9k_hw_startpcureceive(sc->sc_ah,       (sc->sc_flags       &
SC_OP_OFFCHANNEL));
+
+       spin_unlock_bh(&sc->rx.rxbuflock);
 }

 static void ath_edma_stop_recv(struct ath_softc *sc)
@@ -494,10 +494,11 @@ int ath_startrecv(struct ath_softc *sc)
        ath9k_hw_rxena(ah);

 start_recv:
-       spin_unlock_bh(&sc->rx.rxbuflock);
        ath_opmode_init(sc);
        ath9k_hw_startpcureceive(ah,          (sc->sc_flags          &
SC_OP_OFFCHANNEL));

+       spin_unlock_bh(&sc->rx.rxbuflock);
+
        return 0;
 }
```

Рис.9. Описание изменения 952ef78, исправляющего типовую ошибку, проявляющуюся при параллельном выполнении

3. Результаты анализа изменений в драйверах ОС Linux

3.1. Определение ветки ядра, наиболее удобной для анализа изменений

Для анализа изменений в драйверах в рамках данной работы был выбран официальный репозиторий ядра ОС Linux, в котором собраны все версии оригинального ядра за последние несколько лет [26]. На первом этапе был проанализирован ряд последовательных изменений на пути разработки между ядром версии 2.6.38 и версии 2.6.39-rc1. Сводные результаты представлены в Табл. 1. Всего было проанализировано 366 изменений, из которых 100 были исправлениями ошибок. 69 из них (около 19% от общего числа) составляли исправления типовых ошибок.

Изменения (366)		
Расширение функциональности (266 - 73%)	Исправление ошибок (100 – 27%)	
-	Нетиповые ошибки (31 – 8%)	Типовые ошибки (69 – 19%)

Табл. 1. Анализ ряда изменений между ядрами версий 2.6.38 и 2.6.39

Аналогичным образом были проанализированы последовательные изменения, сделанные для ядра 3.0.8 по сравнению с версией 3.0.7 (Табл. 2)[2]. В данном случае оказалось, что около 37% изменений являются исправлениями типовых ошибок, что существенно больше по сравнению с результатами предыдущего анализа (Табл. 1). Это наглядным образом демонстрирует утверждение о том, что при разработке следующей версии ядра большая часть изменений связана с реализацией поддержки новой функциональности, а не с исправлением ошибок. Также стоит отметить, что подряд идущие изменения в стабильной ветке оказались существенно разнообразнее в плане затрагиваемых подсистем по сравнению с основной веткой разработки ядра. В связи с этим в ходе основного анализа, выполненного в рамках данной работы, рассматривались только изменения в стабильные ветки ядра.

Изменения (43)		
Расширение функциональности (17 - 40%)	Исправление ошибок (26 – 60%)	
-	Нетиповые ошибки (10 – 23%)	Типовые ошибки (16 – 37%)

Табл. 2. Анализ изменений стабильной ветки ядра между версиями 3.0.7 и 3.0.8

3.2. Основной анализ изменений в драйверах ядра ОС Linux

В рамках основного анализа рассматривались изменения, которые были сделаны в стабильных ядрах с 2.6.35 по 3.0 в репозитории [26] за время, начиная с 26 октября 2010 года по 26 октября 2011 года. Всего таких изменений насчитывается 3101. Среди данных изменений некоторые являются дубликатами, поскольку одни и те же изменения попадают в различные стабильные ветки. Уникальных изменений за указанный период времени насчитывается 2623 (около 84.6% от общего числа изменений).

[2] Напомним, что нумерация для ядер версии больше 3 отличается, поэтому, например, версия 3.0.7 соответствует версии 2.6.38.7.

Для того чтобы отделить изменения в драйверах от остальных изменений в ядре, рассматривались только такие изменения, которые затрагивали файлы из папок: *crypto*, *drivers*, *sound*, *security*, *include/acpi*, *include/crypto*, *include/drm*, *include/media*, *include/mtd*, *include/pcmcia*, *include/rdma*, *include/rxrpc*, *include/scsi*, *include/sound* и *include/video* (в соответствии с, например, [27] и [28]). Из всех уникальных изменений, сделанных в стабильных ядрах с 26.10.10 по 26.10.11, в драйверах оказалось 1503, т.е. около 57.3% от общего числа уникальных изменений. Сводные результаты основного анализа изменений в драйверах представлены в Табл. 3. Детали анализа доступны публично [29].

Изменения в драйверах (1503)		
Расширение функциональности (321 - 21%)	Исправление ошибок (1182 – 79%)	
-	Нетиповые ошибки (786 – 52%)	Типовые ошибки (396 – 27%)

Табл. 3. Анализ изменений в драйверах стабильных версий ядра с 2.6.35 по 3.0 за время с 26.10.10 по 26.10.11

В соответствии с методикой, предложенной в данной работе, была произведена классификация выявленных типовых ошибок. При классификации не рассматривались изменения, которые являются исправлением типовых специфичных ошибок, связанных с некорректным использованием интерфейса группы драйверов (51 изменение). Изменениям, которые включали исправления сразу нескольких ошибок, приписывалось соответствующее количество классов.

Описание определенных подклассов первого уровня для типовых ошибок в драйверах ОС Linux представлено в Табл. 4. Подклассы ошибок первого уровня с небольшим количеством представителей (не более 3) были объединены в *misc*.

№	Класс	1-й подкласс	Краткое описание типовых ошибок
1	specific	resource	Утечки и использование после освобождения специфичных объектов
2		check_params	Нарушение ограничений на входные параметры
3		context	Вызовы функций в недопустимых контекстах
4		uninit	Некорректная инициализация специфичных объектов
5		lock	Некорректное использование механизмов синхронизации (не требуется параллельное выполнение)
6		style	Нарушение стиля оформления драйверов ядра ОС Linux
7		net	Сетевая подсистема
8		usb	USB подсистема
9		check_ret_val	Отсутствие проверок возвращаемых значений
10		dma	Подсистема прямого доступа к памяти
11		device	Общая модель драйвера
12		misc	Разное
13	generic	null_ptr_deref	Разыменование нулевого указателя
14		resource	Утечки памяти и ее использование после освобождения
15		syntax	Синтаксис
16		int_overflow	Переполнение целых чисел
17		buffer_overflow	Переполнение массива
18		uninit	Использование неинициализованных областей памяти
19		misc	Разное
20	sync	race	Состояние гонки
21		deadlock	Взаимная блокировка

Табл. 4. Описание подклассов первого уровня для типовых ошибок в драйверах ОС Linux

Максимальный уровень вложенности подклассов в рамках проведенной классификации составил 5 подклассов. В среднем насчитывалось 2-3 подкласса. В Табл. 5 приводится описание подклассов второго уровня для типовых специфичных ошибок в драйверах ОС Linux, связанных с

использованием особого контекста. Подклассы для других типовых ошибок составлялись аналогичным образом.

№	Класс и 1-й подкласс	2-й подкласс	Краткое описание типовых ошибок
1	specific:context	sleep	Вызов функций, которые могут заснуть, в контексте прерывания, блокировки и т.д.
2		interrupt	Некорректные операции в контексте прерывания
3		lock	Некорректные операции в контексте блокировки
4		timer	Некорректные операции в контексте таймера

Табл. 5. Описание подклассов второго уровня для типовых ошибок в драйверах ОС Linux, связанных с некорректным использованием механизмов синхронизации

В Табл. 6 и Табл. 7 представлены результаты классификации и распределение типовых ошибок в драйверах ОС Linux с точки зрения различных аспектов. В Табл. 6 приводится общее распределение типовых ошибок по классам и первым подклассам. Цветом выделяются границы, когда суммарное количество типовых ошибок достигают 50% и 80% от общего числа ошибок. В Табл. 7 приводится распределение типовых ошибок по классам и по первым подклассам для каждого класса. По аналогии для каждого класса цветом выделены границы 50% и 80%.

№	Класс и 1-й подкласс	Количество	% от общего	Суммарный % от общего
1	sync:race	60	17.2%	17.2%
2	specific:resource	32	9.2%	26.4%
3	generic:null_ptr_deref	31	8.9%	35.2%
4	specific:check_params	25	7.2%	42.4%
5	generic:resource	24	6.9%	49.3%
6	specific:context	19	5.4%	54.7%
7	specific:uninit	17	4.9%	59.6%
8	generic:syntax	14	4.0%	63.6%
9	specific:lock	12	3.4%	67.0%
10	sync:deadlock	11	3.2%	70.2%
11	specific:style	10	2.9%	73.1%
12	specific:net	10	2.9%	75.9%
13	specific:usb	9	2.6%	78.5%
14	generic:int_overflow	8	2.3%	80.8%

15	generic:buffer_overflow	8	2.3%	83.1%
16	specific:check_ret_val	7	2.0%	85.1%
17	generic:uninit	6	1.7%	86.8%
18	specific:dma	4	1.1%	88.0%
19	specific:device	4	1.1%	89.1%
20	specific:misc	27	7.7%	96.8%
21	generic:misc	11	3.2%	100.0%

Табл. 6. Классификация и распределение типовых ошибок в драйверах ОС Linux

№	Класс	1-й подкласс	Количество	% от общего	Суммарный % от общего
1	**specific** (176 – 50.4%)	resource	32	18.2%	18.2%
2		check_params	25	14.2%	32.4%
3		context	19	10.8%	43.2%
4		uninit	17	9.7%	52.8%
5		lock	12	6.8%	59.7%
6		style	10	5.7%	65.3%
7		net	10	5.7%	71.0%
8		usb	9	5.1%	76.1%
9		check_ret_val	7	4.0%	80.1%
10		dma	4	2.3%	82.4%
11		device	4	2.3%	84.7%
12		misc	27	15.3%	100.0%
13	**generic** (102 – 29.2%)	null_ptr_deref	31	30.4%	30.4%
14		resource	24	23.5%	53.9%
15		syntax	14	13.7%	67.6%
16		int_overflow	8	7.8%	75.5%
17		buffer_overflow	8	7.8%	83.3%
18		uninit	6	5.9%	89.2%
19		misc	11	10.8%	100.0%
20	**sync** (71 – 20.4%)	race	60	84.5%	84.5%
21		deadlock	11	15.5%	100.0%

Табл. 7. Классификация и распределение типовых ошибок в драйверах ОС Linux в зависимости от класса

3.3. Выводы по результатам анализа

На основании результатов, полученных в рамках данного исследования можно сделать следующие выводы:

- Порядка 27% от общего числа уникальных изменений в драйверах являются исправлениями типовых ошибок, 52% - исправления нетиповых ошибок и 21% - реализация поддержки новой функциональности.
- Типовые ошибки в драйверах ядра ОС Linux относятся к одному из 3 классов и 21 подклассов первого уровня. Классы разделяют ошибки общие, специфичные и связанные с параллельным выполнением. Их подклассы соответствуют наиболее выразительным особенностям типовых ошибок и достаточно четко определены. Подклассы следующих уровней в основном показывают, к каким подсистемам ядра относятся ошибки.
- Более 50% типовых ошибок соответствует 5 подклассам первого уровня. Самые часто встречающиеся ошибки связаны с состоянием гонок при параллельном выполнении программы (около 17%). На втором и третьем месте идут соответственно утечки специфичных объектов и разыменования нулевого указателя (по 9%). Более 80% типовых ошибок соответствует 14 подклассам первого уровня.
- Специфичные ошибки составляют примерно половину всех типовых ошибок и распределены по 12 подклассам первого уровня. Общие и связанные с параллельным выполнением программы – 30% и 7 подклассов, и 20% и 2 подкласса соответственно.
- В целом, результаты согласуются с ожиданиями. По заявлениям экспертов ядра ОС Linux именно типовые ошибки, связанные с состоянием гонки и взаимной блокировки (класс *sync*), на сегодняшний день являются наиболее критическими и сложными для обнаружения, вследствие чего их стремятся исправить как можно быстрее. Помимо них, к наиболее важным ошибкам относят разыменование нулевого указателя (класс *generic:null_ptr_deref*), выход за границу массива (класс *generic:buffer_overflow*) и работу с неинициализированными данными (классы *generic:uninit* и *specific:uninit*).
- Полученные результаты не являются полностью достоверными, поскольку часто определить причину ошибки бывает достаточно трудно. Например, некоторые ошибки *generic:null_ptr_deref* могут быть *generic:uninit*, так как разыменование нулевого указателя происходит вследствие того, что данные не были инициализированы, а в соответствующей области памяти оказалось нулевое значение. Также некоторые типовые общие ошибки на самом деле могут быть типовыми специфичными ошибками, так как общее проявление

может быть обусловлено некорректным использованием интерфейса основной части ядра.

- Следует сделать поправку на специфику формирования стабильных веток ядра ОС Linux. Изменение из оригинальной ветки ядра попадает в стабильную, если автор изменения, лицо, проводившее инспекцию кода, или ответственный за соответствующую подсистему посчитает его достаточно важным и пометит, как требующее рассмотрения для включения в стабильную ветку. При этом большинство авторов изменений считают, что предлагать изменение для включения в стабильную ветку не входит в область их ответственности. С другой стороны, известно, что ответственные за целый ряд подсистем ядра, например, wireless или scsi, также не уделяют внимание необходимости помечать изменения. Как следствие возникает неоднородность в составе изменений, попадающих в стабильные ветки ядра, что могло повлиять на результаты анализа. Также следует отметить, что изменения в драйверах интересны меньшему кругу лиц, и поэтому имеют меньшие шансы попасть в стабильные ветки ядра.

4. Аналогичные исследования

В большом количестве работ ошибки в исходном коде ядра ОС Linux, в том числе, в драйверах, были получены на основании применения различных подходов автоматизированного анализа. Например, в статьях [30] и [31] предложены подходы для автоматического выявления неявных правил, которым должен удовлетворять исходный код анализируемых программ, на основе статистических данных. В отличие от [30], где правила получались только для шаблонов из пар элементов (например, если вызвана некоторая функция, то должна быть вызвана и другая), [31] описывает более общий подход, который затрагивает помимо функций переменные и типы и может выявлять правила для большего количества элементов без каких-либо шаблонов. Применительно к ядру ОС Linux подходы позволили выявить тысячи правил и десятки ошибок. Однако, данные подходы не могут гарантировать, что найденные с их помощью ошибки не являются ложными срабатываниями. Они используют множество эвристик, в особенности, при различных статистических оценках и ранжировании выявленных правил. Поэтому требуется существенный ручной труд по анализу обнаруженных ошибок. Кроме того, подходы не могут выявить все возможные правила и соответствующие им ошибки (в [30] задается несколько фиксированных шаблонов; ни в [30], ни в [31] не сообщается о классах и распределении найденных ошибок), что в принципе не позволяет получить полную классификацию и распределение ошибок даже после их ручного анализа.

В статьях [10] и [13] распределение типовых ошибок в ядре ОС Linux было получено автоматизированным образом с помощью инструментов статического анализа на основе предопределенной классификации. Благодаря

данным исследованиям, в частности, было выявлено, что драйверы содержат до 85% всех ошибок, после чего много усилий было потрачено для повышения их надежности. Данным подходам, также как и подходам, описанным ранее, присуще наличие ложных срабатываний. По этой причине результаты, полученные для некоторых классов ошибок, вообще не принимались во внимание. Кроме того, подходы в значительной степени зависят от того, как инструменты статического анализа проверяют соответствующие классы. В [10] и [13] были использованы различные инструменты статического анализа. Очень вероятно, что именно это послужило причиной значительного расхождения полученных результатов.

Результаты, полученные в рамках данной работы, существенно отличаются от результатов данных исследований. Прежде всего, это связано с тем, что в предложенном подходе типовые ошибки анализируются вручную на основе изменений, вносимых в драйверы ОС Linux, в то время, как в аналогичных подходах ошибки определяются автоматизированным образом на основе некоторых версий исходного кода ядра. Автоматизированные решения позволяют проанализировать существенные объемы исходного кода различных версий ядра за сравнительно небольшое время. Поэтому на практике предпочтение отдается автоматизированным подходам. Кроме того, они являются объективными с точностью до реализации эвристик в используемых инструментах анализа. Ручной подход позволяет проанализировать типовые ошибки в драйверах более тщательно. Поэтому на основании его результатов можно получать достаточно полную классификацию и более точное распределение ошибок. Предложенная методика позволяет анализировать ошибки, проявляющиеся при различных условиях (при использовании ОС Linux, при целенаправленном ручном и автоматизированном поиске ошибок и т.д.) за счет анализа изменений в драйверах, сделанных по мере их разработки.

5. Заключение

В статье исследованы возможные пути по выявлению и классификации типовых ошибок в драйверах операционной системы Linux. Показывается, что анализ изменений, вносимых в драйверы ядра Linux, обладает рядом преимуществ по сравнению с другими источниками.

Для анализа изменений в драйверах ОС Linux предложена методика. Данная методика позволяет отделить исправления типовых ошибок от исправлений нетиповых ошибок и от реализации дополнительной функциональности. Также методика позволяет построить классификацию типовых ошибок в драйверах, причем основной упор делается на причину происхождения ошибки, а не на ее проявление.

В качестве основной области применения предложенной методики были выбраны изменения в стабильных версиях ядра ОС Linux, поскольку процент типовых ошибок среди них наиболее высокий. Были проанализированы

изменения, сделанные в стабильных ядрах с 2.6.35 по 3.0, начиная с 26 октября 2010 года по 26 октября 2011 года. Всего было проанализировано 1503 уникальных изменений в драйверах, из которых 396 оказались исправлениями типовых ошибок. Для них была составлена классификация и определено распределение ошибок по соответствующим классам. На основании полученной картины были сделаны соответствующие выводы.

Подводя итог, можно сказать, что выявлять типовые ошибки на основе изменений в драйверах операционной системы Linux эффективно, поэтому эту работу следует продолжать. Существенно облегчить и ускорить ручную работу по анализу изменений может помочь система, которая позволит анализировать изменения итеративным образом непосредственно по мере их появления в репозитории. Разработчики ядра и драйверов ОС Linux могут поспособствовать проведению классификации путем более четкого определения и описания истинной причины исправляемых ими ошибок.

Литература

[1] J. Corbet, G. Kroah-Hartman, A. McPherson. Linux kernel development. How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It. http://go.linuxfoundation.org/who-writes-linux-2012, 2012.

[2] T. Leemhuis. What's new in Linux 3.3. http://www.h-online.com/open/features/What-s-new-in-Linux-3-3-1466872.html, 2012.

[3] Сайт оригинального ядра ОС Linux. http://www.kernel.org.

[4] J. Corbet. How to Participate in the Linux Community. A Guide To The Kernel Development Process. http://www.linuxfoundation.org/sites/main/files/How-Participate-Linux-Community_0.pdf, 2008.

[5] S. M. Kerner. The Red Hat Enterprise Linux 6 Kernel: What Is It? http://www.serverwatch.com/news/article.php/3880131/The-Red-Hat-Enterprise-Linux-6-Kernel-What-Is-It.htm, 2010.

[6] Ядро дистрибутива openSUSE. http://en.opensuse.org/Kernel.

[7] Ядро дистрибутива Debian. http://wiki.debian.org/DebianKernel.

[8] ОС Linux реального времени. https://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html.

[9] ОС Android. http://developer.android.com/guide/basics/what-is-android.html.

[10] A. Chou, J. Yang, B. Chelf, S. Hallem, and DR Engler. An Empirical Study of Operating System Errors. Proc. 18th ACM Symp. Operating System Principles, 2001.

[11] M. Swift, B. Bershad, H. Levy. Improving the reliability of commodity operating systems. In: SOSP ’03: Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003.

[12] A. Ganapathi, V. Ganapathi, D. Patterson. Windows XP kernel crash analysis. Proceedings of the 2006 Large Installation System Administration Conference, 2006.

[13] N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, and Gilles Muller. Faults in linux: ten years later. Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS '11), USA, 2011.

[14] S. Butt, V. Ganapathy, M.M. Swift, C.-C. Chang. Protecting Commodity Operating System Kernels from Vulnerable Device Drivers. Computer Security Applications Conference (ACSAC '09), 2009.

[15] D. Tian, X. Xiong, C. Hu, P. Liu. Policy-centric protection of OS kernel from vulnerable loadable kernel modules. Proceedings of the 7th international conference on Information security practice and experience, 2011.

[16] E.S. Raymond. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O’Reilly, Sebastopol, CA, USA, 2001.

[17] R.L. Glass. Facts and Fallacies of Software Engineering. AddisonWesley, Professional, Sebastopol, CA, USA, 2003.

[18] ISO/IEC TR 24772. Information Technology — Programming Languages — Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use, 2010.

[19] G. Kroah-Hartman. The Linux kernel driver interface. http://www.kernel.org/doc/Documentation/stable_api_nonsense.txt.

[20] Документация ядра ОС Linux. http://kernel.org/doc/Documentation.

[21] A. Rubini. Linux Device Drivers (Nutshell Handbooks). O'Reilly Media, 1st edition, February 24, 1998.

[22] M. Beck, H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus, D. Verworner. Linux Kernel Internals. Addison-Wesley Professional, 2 edition, December 16, 1997.

[23] Список рассылки ядра ОС Linux. https://lkml.org.

[24] Проиндексированный исходный код ядра ОС Linux различных версий. http://lxr.linux.no.

[25] R. Russell. Unreliable Guide To Hacking The Linux Kernel. http://www.kernel.org/doc/htmldocs/kernel-hacking.html, 2005.

[26] Репозиторий стабильных версий ядра ОС Linux. http://git.kernel.org/?p=linux/kernel/git/stable/linux-stable.git;a=summary.

[27] M. K. Saad. Browsing Linux Kernel. Linux Day, May 6, 2007.

[28] G. Kroah-Hartman. Kernel development statistics for 2.6.35. http://lwn.net/Articles/395961, 2010.

[29] Результаты анализа изменений в драйверах ОС Linux. http://linuxtesting.org/downloads/ldv-commits-analysis-2012.zip.

[30] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: a general approach to inferring errors in systems code. ACM SIGOPS Operating Systems Review, v.35 n.5, December, 2001.

[31] Z. Li and Y. Zhou. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. ACM SIGSOFT Software Engineering Notes, v.30 n.5, September, 2005.

Analysis of typical faults in Linux operating system drivers

Mutilin V.S., Novikov E.M., Khoroshilov A.V.
mutilin@ispras.ru, novikov@ispras.ru, khoroshilov@ispras.ru
ISP RAS, Moscow, Russia

Abstract. Fast evolution of the Linux operating system kernel and drivers, developed by a big programmers community distributed all over the world, led to nowadays there is not a common base of rules that completely describe a correct interaction between drivers and the kernel. This is an obstacle both for programmers that do not have expert knowledge in all peculiarities of the given interaction, and for development and application of tools that could find corresponding typical faults in the automatic way. The given paper presents a method to detect and to classify typical faults and corresponding rules. This method is based on analysis of changes, made to Linux operating system drivers.

The paper gives results of the method application to stable versions of the Linux kernel from 2.6.35 till 3.0 starting from October 26, 2010 till October 26, 2011. We analyzed in total 1503 unique commits to drivers, marked 396 (about 27%) of them as fixes of typical faults and provided a classification and a distribution by classes for these typical faults. We distinguished 3 classes of typical faults: generic (faults all C programs are subjected to), specific (faults related to misuses of the Linux kernel API) and synchronization (faults related to parallel execution). We found that specific faults constitute about 50% of all typical faults. Then each typical fault was ascribed to one of 21 first-level subclasses: 12, 7 and 2 for specific, generic and synchronization classes correspondingly. We found that more than 50% of typical faults correspond to 5 first-level subclasses. More than 80% of typical faults correspond to 14 first-level subclasses. The most frequent faults were race conditions during parallel execution – they account for about 17% of all typical faults. Second and third places with about 9% were occupied by leaks of specific resources and null pointer deference.

Keywords: operating system; kernel; driver; interaction rule; faults classification.

References

[1]. Corbet J., Kroah-Hartman G., McPherson A. Linux kernel development. How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It. http://go.linuxfoundation.org/who-writes-linux-2012, 2012.
[2]. Leemhuis T. What's new in Linux 3.3. http://www.h-online.com/open/features/What-s-new-in-Linux-3-3-1466872.html, 2012.
[3]. Vanilla Linux kernel. http://www.kernel.org.
[4]. Corbet J. How to Participate in the Linux Community. A Guide To The Kernel Development Process. http://www.linuxfoundation.org/sites/main/files/How-Participate-Linux-Community_0.pdf, 2008.
[5]. Kerner S.M. The Red Hat Enterprise Linux 6 Kernel: What Is It? http://www.serverwatch.com/news/article.php/3880131/The-Red-Hat-Enterprise-Linux-6-Kernel-What-Is-It.htm, 2010.
[6]. openSUSE kernel. http://en.opensuse.org/Kernel.
[7]. Debian kernel. http://wiki.debian.org/DebianKernel.

[8]. Real-Time Linux. https://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html.
[9]. Android operating system. http://developer.android.com/guide/basics/what-is-android.html.
[10]. Chou A., Yang J., Chelf B., Hallem S., Engler D. An Empirical Study of Operating System Errors. In Proc. 18th ACM Symposium on Operating Systems Principles (SOSP), pp. 73-88, 2001. doi: 10.1145/502034.502042
[11]. Swift M., Bershad B., Levy H. Improving the reliability of commodity operating systems. In Proc. 19th ACM Symposium on Operating Systems Principles (SOSP), pp. 73-88, 2003. doi: 10.1145/502034.502042
[12]. Ganapathi A., Ganapathi V., Patterson D. Windows XP kernel crash analysis. In Proc. 20th Conference on Large Installation System Administration (LISA), pp. 149-159, 2006.
[13]. Palix N., Thomas G., Saha S., Calves C., Lawall J., Muller G. Faults in linux: ten years later. In Proc. 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 305-318, 2011. doi: 10.1145/1950365.1950401
[14]. Butt S., Ganapathy V., Swift M.M., Chang C.-C. Protecting Commodity Operating System Kernels from Vulnerable Device Drivers. In Proc. Computer Security Applications Conference (ACSAC), pp. 301-310, 2009. doi: 10.1109/ACSAC.2009.35
[15]. Tian D., Xiong X., Hu C., Liu P. Policy-centric protection of OS kernel from vulnerable loadable kernel modules. In Proc. Information Security Practice and Experience, LNCS, vol. 6672, pp. 317-332, 2011. doi: 10.1007/978-3-642-21031-0_24
[16]. Raymond E.S. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media, 1999.
[17]. Glass R.L. Facts and Fallacies of Software Engineering. Addison Wesley, 2002.
[18]. ISO/IEC TR 24772. Information Technology — Programming Languages — Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use, 2010.
[19]. Kroah-Hartman G.. The Linux kernel driver interface. http://www.kernel.org/doc/Documentation/stable_api_nonsense.txt.
[20]. Linux kernel documentation. http://kernel.org/doc/Documentation.
[21]. Rubini A. Linux Device Drivers (Nutshell Handbooks). O'Reilly Media, 1998.
[22]. Beck M., Bohme H., Dziadzka M., Kunitz U., Magnus R., Verworner D. Linux Kernel Internals. Addison-Wesley, 1996.
[23]. Linux kernel mailing list. https://lkml.org.
[24]. Indexed source code of various versions of the Linux kernel. http://lxr.linux.no.
[25]. Russell R. Unreliable Guide To Hacking The Linux Kernel. http://www.kernel.org/doc/htmldocs/kernel-hacking.html, 2005.
[26]. Repository of stable versions of the Linux kernel. http://git.kernel.org/?p=linux/kernel/git/stable/linux-stable.git;a=summary.
[27]. Saad M.K. Browsing Linux Kernel. Linux Day, 2007.
[28]. Kroah-Hartman G. Kernel development statistics for 2.6.35. http://lwn.net/Articles/395961, 2010.
[29]. Results of analysis of commits to Linux drivers. http://linuxtesting.org/downloads/ldv-commits-analysis-2012.zip.
[30]. Engler D., Chen D.Y., Hallem S., Chou A., Chelf B. Bugs as deviant behavior: a general approach to inferring errors in systems code. In Proc. 18th ACM Symposium on Operating Systems Principles (SOSP), vol. 35, issue 5, pp. 57-72, 2001. doi: 10.1145/502034.502041

[31]. Li Z., Zhou Y. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. In Proc. 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE), vol. 30, issue 5, pp. 306-315, 2005. doi:10.1145/1081706.1081755