Федеральное государственное бюджетное учреждение науки
Институт системного программирования Российской академии наук

Верификация драйверов операционной системы Linux при помощи предикатных абстракций

Мутилин Вадим Сергеевич

Научный руководитель:
д. ф.-м. н., проф. Петренко Александр Константинович

Москва, 2012

Статистика ядра Linux (1)

ИСП РАН

- Более 1000 активных разработчиков [1]

Версия ядра	Количество разработчиков	Количество компаний
2.6.11	389	68
2.6.12	566	90
...		
2.6.38	1,198	220
2.6.39	1,258	239
3.0	1,131	331
3.1	1,168	212
3.2	1,316	226

[1] Kroah-Hartman G, Corbet J, McPherson A (2012) Linux Kernel Development, http://go.linuxfoundation.org/who-writes-linux-2012

- 4.3 патчей в час (в среднем за 7 лет) [1]

Версия ядра	Дата выпуска	Изменений в час
2.6.11	02.03.2005	2.18
2.6.12	17.05.2005	1.95
...		
2.6.38	14.03.2011	5.78
2.6.39	18.05.2011	6.58
3.0	21.07.2011	5.96
3.1	24.10.2011	3.81
3.2	04.01.2012	6.88

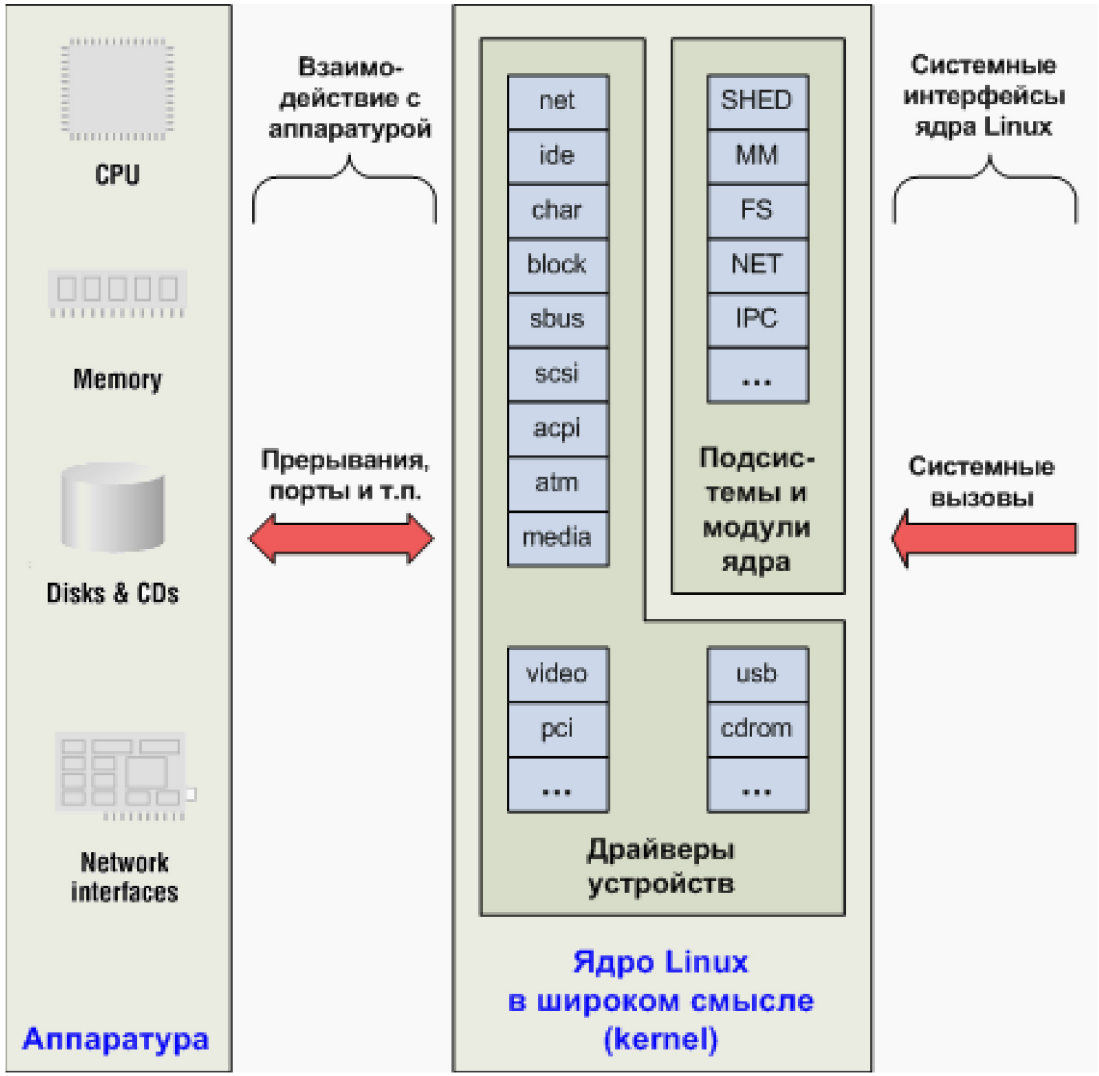
[1] Kroah-Hartman G, Corbet J, McPherson A (2012) Linux Kernel Development, http://go.linuxfoundation.org/who-writes-linux-2012

ИСП РАН

- более 15 млн. строк кода [1]

Версия ядра	Файлов	Строк кода
2.6.11	17,090	6,624,076
2.6.12	17,360	6,777,860
...		
2.6.38	36,868	14,211,814
2.6.39	36,713	14,537,764
3.0	36,788	14,651,135
3.1	37,095	14,776,002
3.2	37,726	15,004,006

[1] Kroah-Hartman G, Corbet J, McPherson A (2012) Linux Kernel Development, http://go.linuxfoundation.org/who-writes-linux-2012


CPU
Memory
Disks & CDs
Network interfaces
Аппаратура
Взаимо-действие с аппаратурой
Прерывания, порты и т.п.
net
ide
char
block
sbus
scsi
acpi
atm
media
video
pci
...
usb
cdrom
...
Драйверы устройств
SHED
MM
FS
NET
IPC
...
Подсис-темы и модули ядра
Ядро Linux в широком смысле (kernel)
Системные интерфейсы ядра Linux
Системные вызовы

- Составляют до 70% исходных кодов ядра
- 85% падений происходит из-за драйверов

1 A. Chou et al. (2001) An Empirical Study of Operating System Errors, In Proc. of the 18-th ACM Symp. Operating System Principles, ACM Press.

2 Swift M, Bershad B, Levy H. (2003) Improving the reliability of commodity operating systems. In SOSP'03: Symposium on Operating System Principles.

с версии 2.6.35 по 3.0

за время с 26.10.2010 по 26.10.2011

<table>
<tr><td colspan="3">Изменения в драйверах (1503)</td></tr>
<tr><td rowspan="2">Расширение функциональности (321 – 21%)</td><td colspan="2">Исправление ошибок (1182 – 79%)</td></tr>
<tr><td>Нетиповые ошибки (786 – 67%)</td><td>Типовые ошибки (396 – 33%)</td></tr>
</table>

Класс	Количество	% от общего
Специфичные	176	50.4%
Общие	102	29.2%
Связанные с параллельным выполнением	71	20.4%
Всего	349	100%

Всего 349 типовых ошибок за вычетом 47 типовых специфичных ошибок, связанных с некорректным использованием интерфейса группы драйверов

Подходы к обеспечению качества

ИСП РАН

- Тестирование (динамический анализ)
- Общецелевые методы статического анализа
- Высокоточные методы статической верификации

- Большинство драйверов публикуется вместе с исходным кодом
- Небольшой размер драйверов (в среднем 2-3 тыс. строк)
- Практически нет арифметики с плавающей запятой
- Немного рекурсивных функций

```
int main(int argc,char* argv[])
{
  ...
 other_func(var);
  ...
}
```

```
void other_func(int v)
{
   ...
   assert( x != NULL);
}
```

ИСП РАН

```
static struct pci_driver DAC960_pci_driver = {
        .name           = "DAC960",
        .id_table       = DAC960_id_table,
        .probe          = DAC960_Probe,
        .remove         = DAC960_Remove,
};

static int DAC960_init_module(void)
{
        int ret;
        ret = pci_register_driver(&DAC960_pci_driver)
#ifdef DAC960_GAM_MINOR
        if (!ret)
                DAC960_gam_init();
#endif
        return ret;
}
...
module_init(DAC960_init_module);
module_exit(DAC960_cleanup_module);
```

Регистрация функций обработчиков

Нет явных вызовов функций инициализации и выхода

This is being written to try to explain why Linux does not have a binary kernel interface, nor does it have a stable kernel interface. Please realize that this article describes the in kernel interfaces, not the kernel to userspace interfaces. The kernel to userspace interface is the one that application programs use, the syscall interface. That interface is very stable over time, and will not break.

Executive Summary

You think you want a stable kernel interface, but you really do not, and you don't even know it. What you want is a stable running driver, and you get that only if your driver is in the main kernel tree. You also get lots of other good benefits if your driver is in the main kernel tree, all of which has made Linux into such a strong, stable, and mature operating system which is the reason you are using it in the first place.

Greg Kroah-Hartman

Сравнение существующих систем верификации драйверов

ИСП РАН

Требования	**Microsoft SDV**	**Avinux**	**DDVerify**
Переиспользование информации о параметрах сборки	+	±	–
Генерация модели окружения	По аннотации	Ручная	Для 3-х типов драйверов
Поддержка добавления новых правил корректности	+	+	±
Сопровождаемость в условиях непрерывного развития ядра	Не требуется	±	–
Визуализация результата и трассы ошибки	+	–	±
Поддержка интеграции новых инструментов верификации	–	–	–

Цель работы – разработка метода верификации драйверов устройств операционных систем для проверки выполнения правил корректного взаимодействия драйверов с ядром операционной системы.

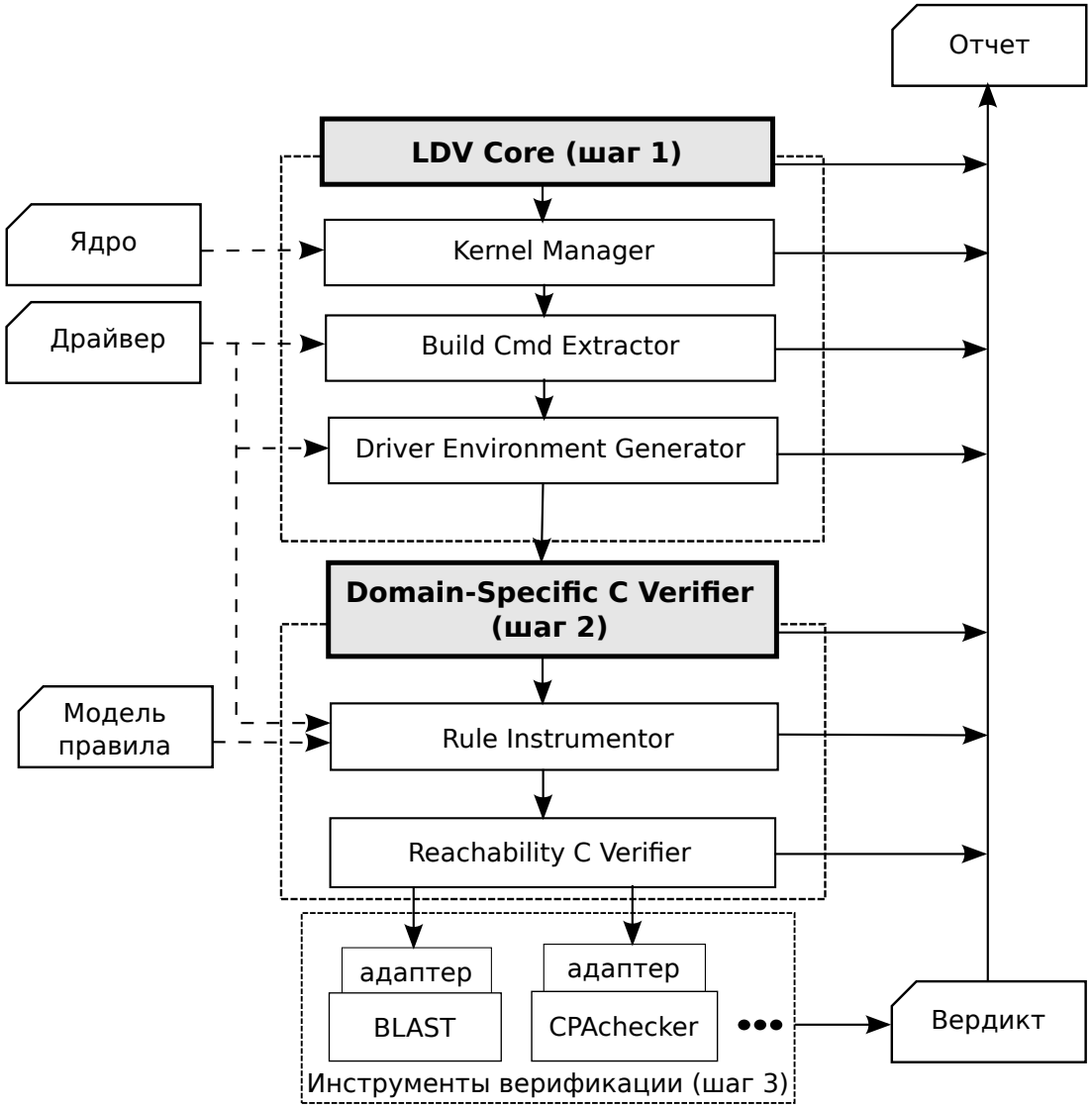
Задачи работы

ИСП РАН

1. Провести анализ существующих методов верификации драйверов;
2. Провести анализ ошибок в драйверах ОС Linux, приводящих к некорректному взаимодействую с ядром ОС;
3. Разработать метод верификации и архитектуру системы верификации драйверов ОС Linux при помощи предикатных абстракций, обеспечивающий:
   - верификацию драйверов в условиях непрерывного развития ядра;
   - возможности расширения (конфигурируемости) системы верификации драйверов за счет пополнения набора правил корректности и набора инструментов верификации.
4. Разработать систему верификации, реализующую метод;
5. Оценить реализацию метода на практике, дать оценку области применимости метода.

Компоненты архитектуры и шаги метода верификации

ИСП РАН



Требования к синтезируемому окружению

ИСП РАН

- Должно воспроизводить те же сценарии взаимодействия с драйвером, что и реальное окружение драйвера в операционной системе
- Конструкция синтезируемого окружения должна удовлетворять требованиям инструментов статической верификации

Структура – набор функций-обработчиков заданных типов (например структура file_operation включает функции open, release, read, write, . . .).

Удовлетворяют ограничениям:

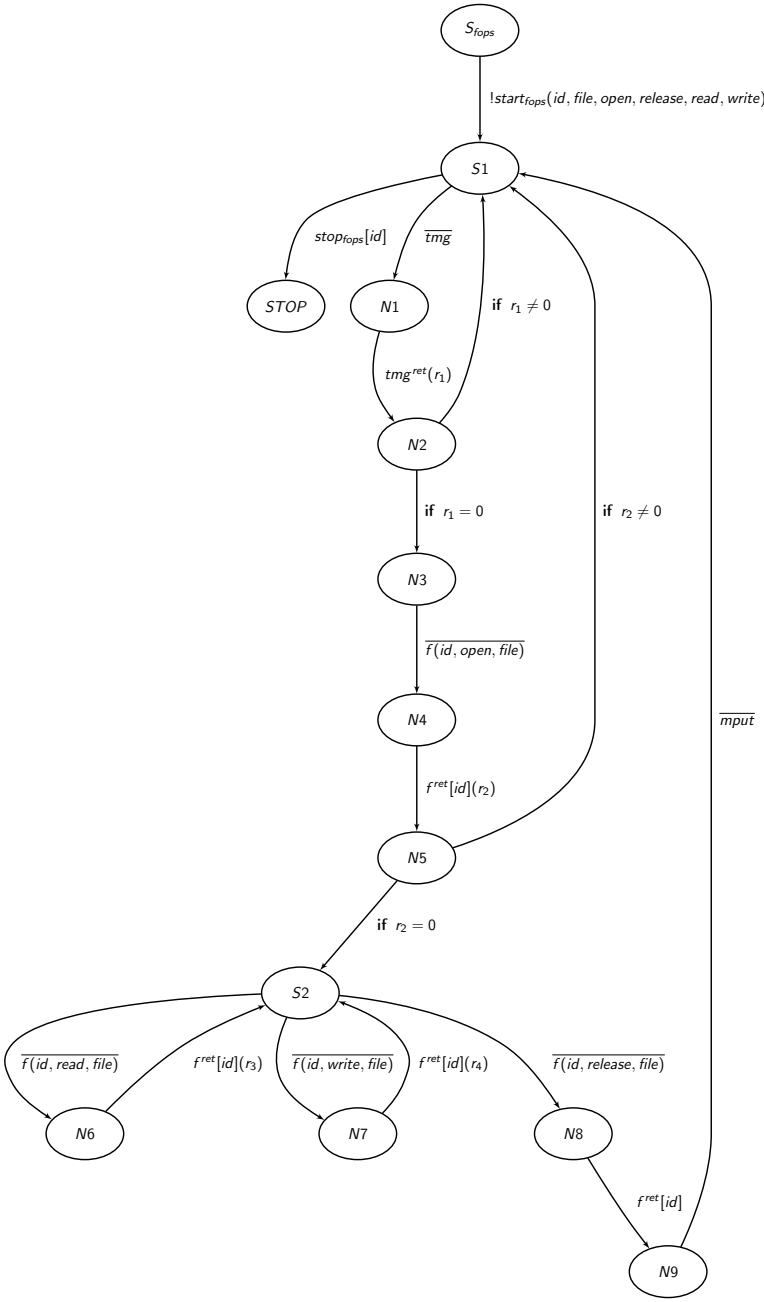
1. на порядок вызова обработчиков одной струкутры;
2. на контекст вызова и передаваемые значения обработчиков структур;
3. на порядок вызова обработчиков нескольких структур;

Определение процессов по “The Polyadic π-Calculus: a Tutorial, Robin Milner, October 1991”.

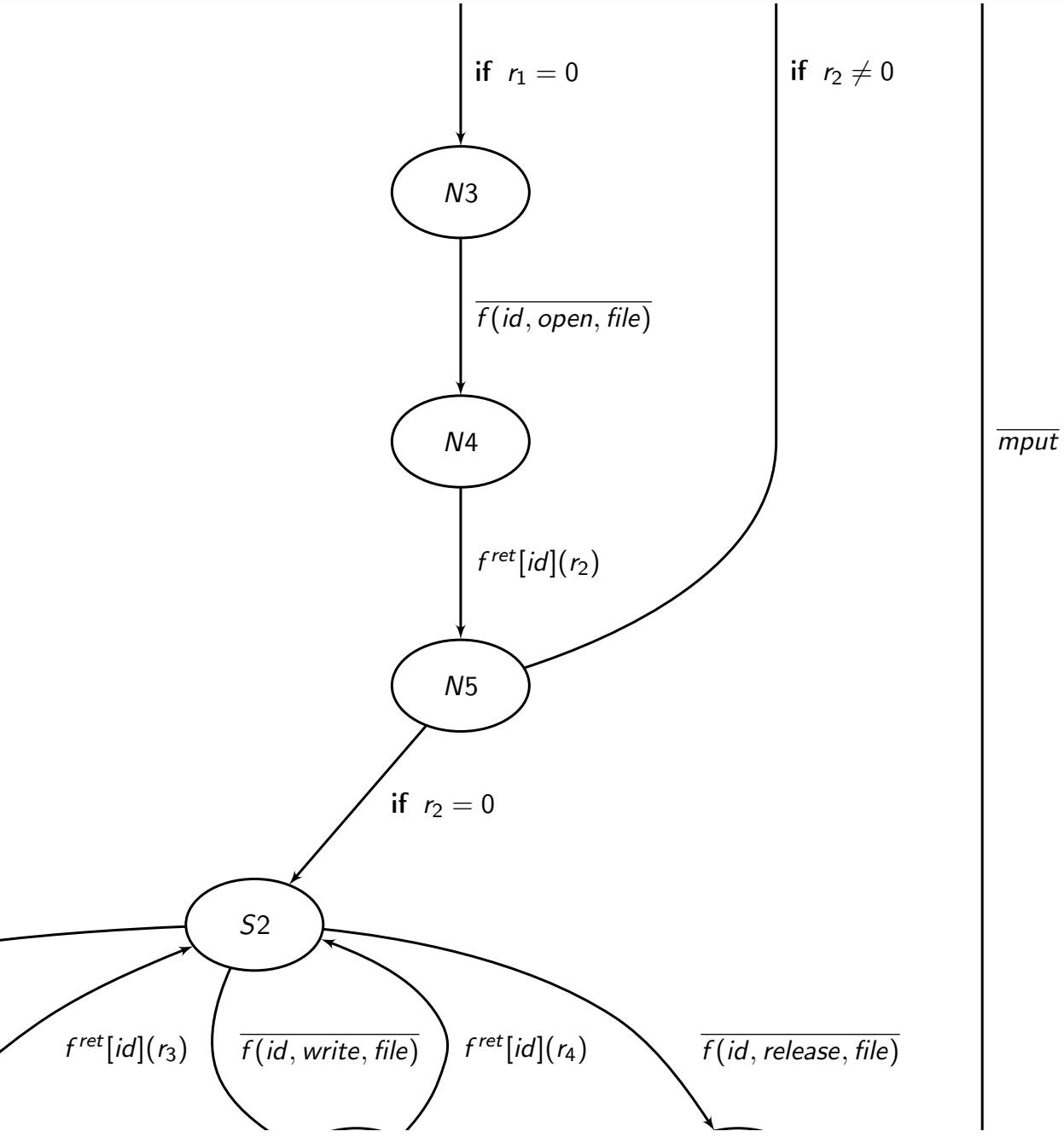
Пример процесса file_operations (S_{fops}):

- $S_{fops} \stackrel{\text{def}}{=} !start_{fops}(id, file, open, release, read, write)$ $.S_1(id, file, open, close, read, write)$
- $S_1(id, file, open, release, read, write) \stackrel{\text{def}}{=}$ $\overline{stop_{fops}[id]}.0 + \overline{tmg}.tmg^{ret}(r_1)$.**if** $r_1 \neq 0$ **then** S_1 **else** $\overline{f(id, open, file)}.f^{ret}[id](r_2)$.**if** $r_2 \neq 0$ **then** S_1 **else** S_2
- $\overline{S_2(id, file, open}, release, read, write) \stackrel{\text{def}}{=}$ $\overline{f(id, read, file)}.f^{ret}[id](r_3).S_2 +$ $\overline{f(id, write, file)}.f^{ret}[id](r_4).S_2 +$ $\overline{f(id, release, file)}.f^{ret}[id].\overline{mput}.mput^{ret}.S_1$

Процесс file_operations (S_{fops})



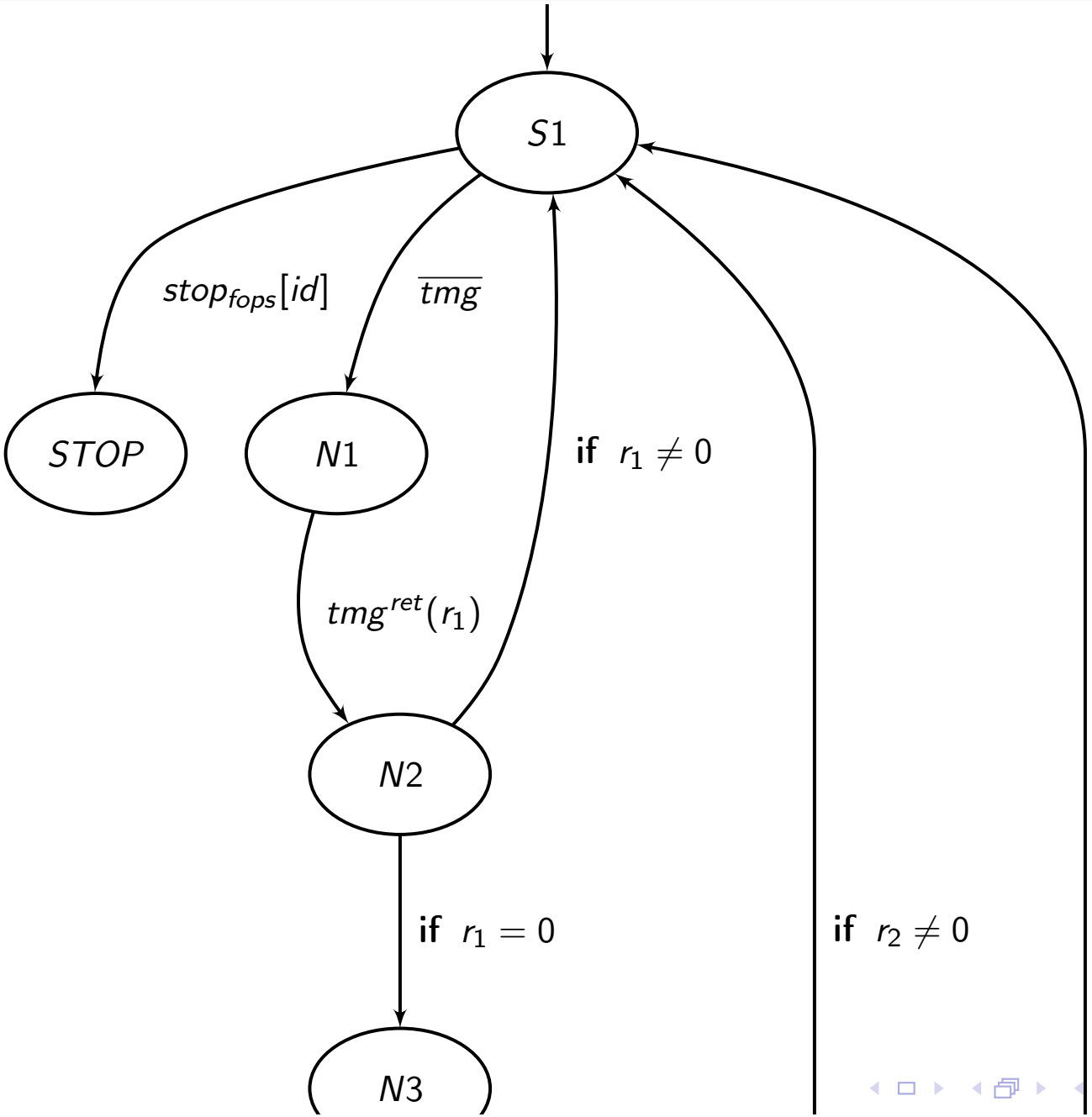
Процесс file_operations (S_{fops})



$\overline{mput}$

Процесс file_operations (S_{fops})

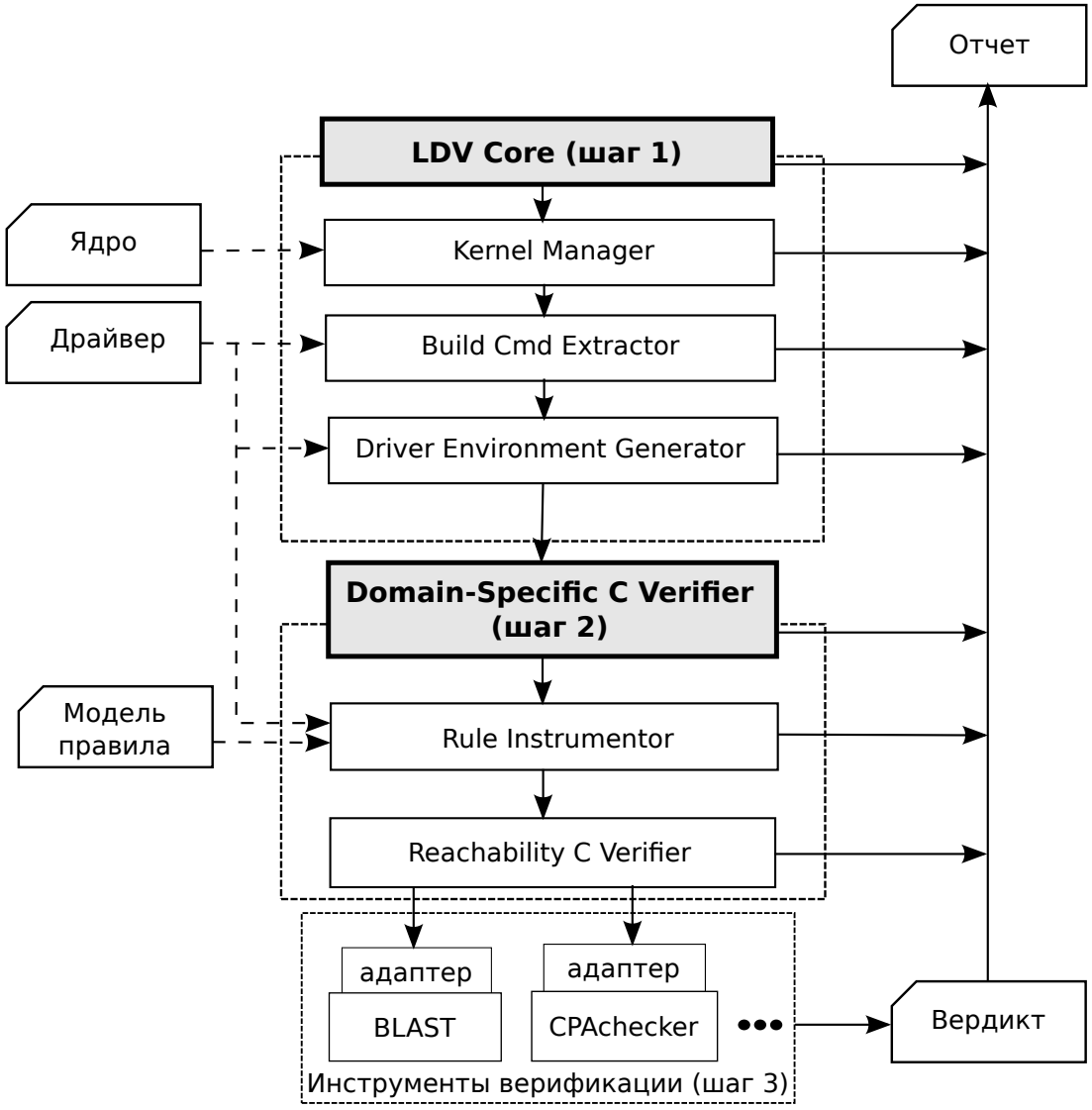
ИСП РАН



```
void main(int argc, char* argv[]) {
  while(true) {
    switch(nondet_int()) {
      case 0: //процесс P_1, действие α_1:
        if(s->state == K) {
          ...
        }
        break;
      case 1: //процесс P_1, действие α_2:
      ...
    }
  }
  break_loop:
}
```

Компоненты архитектуры и шаги метода верификации

ИСП РАН



Пример правила корректности

ID 0077: Выделение памяти с флагом NOIO в контексте usb_lock

ОПИСАНИЕ: Выделение памяти с флагом *GFP_KERNEL* может использовать операции ввода/вывода с устройством хранения, которые могут завершаться с ошибкой, что приводит к необходимости сброса состояния утсройства. Поэтому флаг *GFP_KERNEL* не может использоваться между вызовами *usb_lock_device()* и *usb_unlock_device()*. Вместо него следует использовать флаг *GFP_NOIO*.

ССЫЛКИ:
Пример исправления:
http://git.kernel.org/?p=linux/kernel/git/stable/linux-stable.git;a=commitdiff;h=186c74d336e2c1377df9e5dc88f7966b2dd6acf7

ИСП РАН

```
usb_lock_device(lock);
...
buf = kmalloc(size, GFP_KERNEL); // Bug!
...
usb_unlock_device(lock);
```

Модель правила 0077 (1)

ИСП РАН

В модельном состоянии отражаем захвачена ли USB блокировка.

```
int usblock_held = 0;/*в начале свободна*/
```

Проверка условия правила: если USB блокировка захвачена (spinlock_held == 1), то функции выделения памяти должны вызываться с флагом GFP_NOIO.

```
void ldv_check_alloc_flags(gfp_t flags)
{ if (usblock_held) ldv_assert(flags == GFP_NOIO);}
```

Функция захвата USB блокировки:

```
void ldv_usb_lock_device(void)
{ usblock_held = 1; }
```

Функция освобождения USB блокировки:

```
void ldv_usb_unlock_device(void)
{ usblock_held = 0; }
```

Функции, для которых нужно проверять значение флага

```
pointcut ARG_2: execution(static inline void *kmalloc(..))
        || execution(static inline void *kcalloc(..))
        ...
```

В самой модели можно использовать именованый срез по имени:

```
before: ARG_2
{
  ldv_check_alloc_flags($arg2);
}
```

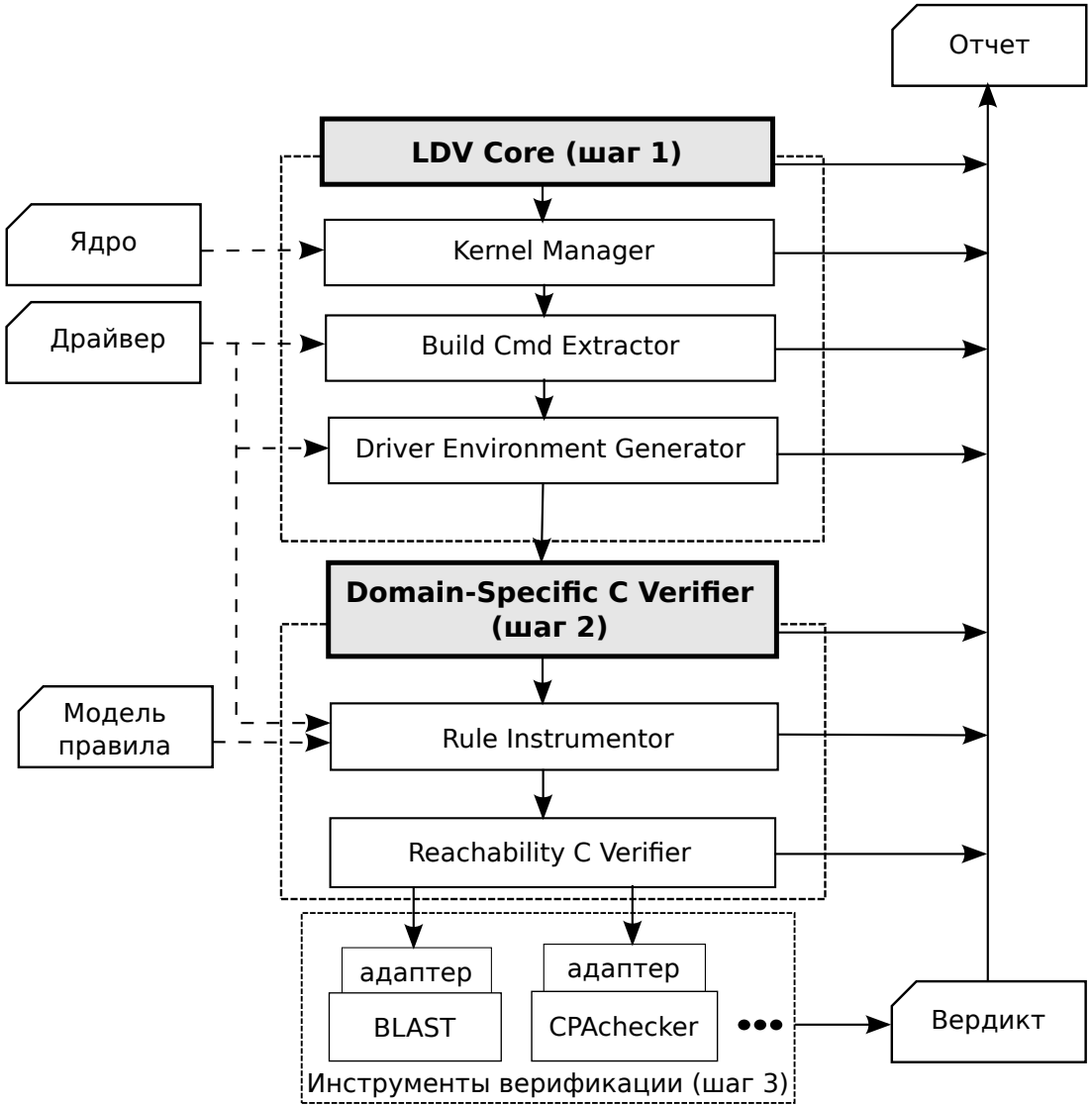
```
usb_lock_device(lock);
...
buf = kmalloc(size, GFP_KERNEL); // Bug!
...
usb_unlock_device(lock);
```

CIF – C Instrumentation Framework
Е.М.Новиков. Подход к реализации аспектно-ориентированного программирования для языка Си. Программирование 2012.

```
ldv_usb_lock_device(lock);
...
buf = aux_kmalloc(size, GFP_KERNEL); // Bug!
...
ldv_usb_unlock_device(lock);
...
static inline void *aux_kmalloc(size_t size, gfp_t flags) {
    ldv_check_alloc_flags(flags);
    return kmalloc(size, flags);
}
```

Компоненты архитектуры и шаги метода верификации

ИСП РАН



Berkeley
Lazy
Abstraction
Software
Verification
Tool

BLAST – инструмент статической вирификации Си программ. Он использует подход, основанный на абстракции и уточнении по контрпримерам (CEGAR) для построения абстрактной модели, на которой проверяются свойства достижимости.

- Настройка автоматического управления памятью в OCaml (20% прироста в количестве посещенных точек программы).
- Исправления в синтаксическом анализаторе CIL (было 100% стало 2% ошибок разбора)
- Была улучшена работа с решателями, получающими на вход формулы в формате SMTLIB
- Оптимизирован алгоритм фильтрации формулы пути (время работы алгоритма стало O(log N), вместо O(N))
- Был реализован новый алгоритм анализа алиасов, который позволяет анализировать программы, использующие указатели на базовые типы и структуры.

Оптимизации в инструменте BLAST 2.7 (2)

ИСП РАН

- old – BLAST 2.5, 2008 (Университет Беркли)
- new – BLAST 2.7, 2012 (ИСП РАН)

Task	Total	Safe	Unsafe	Unknown	In	Ok	Fail	Time	Time Ok	Time Fail	Unn
Task description **old**	2225	1363	8	854	2155	1371	784	381 022,86	10 022,21	371 000,65	
Task description **new**	2230	1750	42	438	2160	1792	368	44 965,25	15 412,07	29 553,18	

+34 2x 8.5x

ИСП РАН

BLAST 2.7

- 1-ое место в категории DeviceDrivers64
- 3-ое место в категории DeviceDrivers32

Competition candidate	**BLAST 2.7**	**CPAchecker-ABE 1.0.10**	**CPAchecker-Memo 1.0.10**	**ESBMC 1.17**	**FShell 1.3**	**LLBMC 0.9**	**Predator 2011-10-11**	**QARMC-HSF**	**SATabs 3.0**	**Wolverine 0.5c**
Representing Jury Member	Vadim Mutilin	Philipp Wendler	Daniel Wonisch	Bernd Fischer	Helmut Veith	Carsten Sinz	Tomas Vojnar	Andrey Rybalchenko	Michael Tautschnig	Georg Weissenbacher
Affiliation	Moscow, Russia	Passau, Germany	Paderborn, Germany	Southampton, UK	Vienna, Austria	Karlsruhe, Germany	Brno, Czechia	Munich, Germany	Oxford, UK	Princeton, USA
ControlFlowInteger 93 files, max score: 144	71 9900 s	141 1000 s	140 3200 s	102 4500 s	28 580 s	100 2400 s	17 1100 s	140 4800 s	75 5400 s	39 580 s
DeviceDrivers 59 files, max score: 103	72 30 s	51 97 s	51 93 s	63 160 s	20 3.5 s	80 1.6 s	80 1.9 s	--	71 140 s	68 65 s
DeviceDrivers64 41 files, max score: 66	55 1400 s	26 1900 s	49 500 s	10 870 s	0 0 s	1 110 s	0 0 s	--	32 3200 s	16 1300 s
HeapManipulation 14 files, max score: 24	--	4 16 s	4 16 s	1 220 s	--	17 210 s	20 1.0 s	--	--	--
SystemC 62 files, max score: 87	33 4000 s	45 1100 s	36 450 s	67 760 s	--	8 2.4 s	21 630 s	8 820 s	57 5000 s	36 1900 s
Concurrency 8 files, max score: 11	--	0 0 s	0 0 s	6 270 s	0 0 s	--	0 0 s	--	1 1.4 s	--
Overall 277 files, max score: 435	231 15000 s	267 4100 s	280 4300 s	249 6800 s	48 580 s	206 2700 s	138 1700 s	148 5600 s	236 14000 s	159 3800 s

Описание подклассов специфичных ошибок взаимодействия в драйверах

ИСП РАН

№	Краткое описание подкласса	Кол-во – %
1	Утечки ресурсов и использование после освобождения	32 – 18.2%
2	Нарушение ограничений на входные параметры	25 – 14.5%
3	Вызовы функций в недопустимых контекстах	19 – 10.8%
4	Некорректная инициализация специфичных объектов	17 – 9.7%
5	Некорректное использование механизмов синхронизации	12 – 6.8%
6	Нарушение стиля оформления драйверов	10 – 5.7%
7	Некорректное использование интерфейса сетевой подсистемы	10 – 5.7%
8	Некорректное использование интерфейса USB подсистемы	9 – 5.1%
9	Отсутствие проверок возвращаемых значений	7 – 4.0%
10	Некорректное использование интерфейса подсистемы прямого доступа к памяти	4 – 2.3%
11	Некорректное использование интерфейса общей модели драйвера	4 – 2.3%
12	Разное	27 – 15.3%

Описание соответствия подклассов специфичных ошибок и реализованных правил

ИСП РАН

№	Краткое описание подкласса	Правила
1	Утечки ресурсов и использование после освобождения	101, 133
2	Нарушение ограничений на входные параметры	100, 103, 129, 142
3	Вызовы функций в недопустимых контекстах	43, 102, 113, 123
4	Некорректная инициализация специфичных объектов	110, 111, 130
5	Некорректное использование механизмов синхронизации	32, 143
6	Нарушение стиля оформления драйверов	107
7	Некорректное использование интерфейса сетевой подсистемы	114, 138
8	Некорректное использование интерфейса USB подсистемы	106, 136
9	Отсутствие проверок возвращаемых значений	134
10	Некорректное использование интерфейса подсистемы прямого доступа к памяти	29
11	Некорректное использование интерфейса общей модели драйвера	132
12	Разное	131

ИСП РАН

- Более 100 описаний правил
- Более 40 моделей правил

Тема
145: Usage of clock functions
144: correct terminations of functions in usb-system
143: Usage of semaphores
73: Function free_irq should be called only once
test_ERR: Testing models of kernel functions from <linux/err.h>
142: Usage of mod_timer()
140: alloc_netdev() must be paired with free_netdev()
140: register_netdev() must be paired with unregister_netdev()
139: napi_enable() must be paired with napi_disable()
138: NAPI context resource leaks - netif_napi_del and netif_napi_add
32_1: Locking a mutex twice or unlocking without prior locking
137: Don't mark EXPORT_SYMBOL functions as __init
010: Usage of a GFP_ATOMIC flag in functions of memory allocation in a context of interrupt.
136: usb_deregister() should be called before usb_serial_deregister() when the device is plugged in
134: Error handling for critical functions in probe()
133: Don't call function kfree_skb twice
132: Usb device reference counting with usb_get_dev/usb_put_dev and interface_to_usbdev

Найденные ошибки

- 75 ошибок уже исправлены (http://linuxtesting.ru/results/ldv)

Номер	Тип	Краткое описание	Добавлено	Принято	Статус
L0075	Блокировка	mpt2sas: двойной захват мьютекса в состоянии NON_BLOCKING	2012-10-02	https://lkml.org/lkml/2011/4/18/331 commit	Исправлено в kernel 3.7-rc1
L0074	Падение	usb: gadget: mv_udc: выделение памяти с флагом GFP_KERNEL в build_dtd() вызываемой из обработчика прерываний	2012-10-01	https://lkml.org/lkml/2012/7/5/567 commit	Исправлено в kernel 3.7-rc1
L0073	Падение	pcmcia: synclink_cs: fix potential tty NULL dereference	2012-10-01	https://lkml.org/lkml/2012/9/13/556 commit	Исправлено в kernel 3.7-rc1
L0072	Падение	staging: sbe-2t3e3: проблемы с обработкой ошибок в t3e3_init_channel()	2012-10-01	https://lkml.org/lkml/2012/9/25/296 commit	Исправлено в kernel 3.7-rc1
L0071	Падение	USB: omninet: разыменование нулевого указателя	2012-10-01	https://lkml.org/lkml/2012/9/13/497 commit	Исправлено в kernel 3.7-rc1
L0070	Утечка	ddbridge: утечка ресурсов в module_init_ddbridge()	2012-10-01	https://lkml.org/lkml/2012/8/15/475 commit	Исправлено в kernel 3.7-rc1
L0069	Утечка	virtio: console: fix error handling in init() function	2012-10-01	https://lkml.org/lkml/2012/9/1/85 commit	Исправлено в kernel 3.7-rc1
L0068	Падение	ppdev: ppdev_init() возвращает 0 в случае ошибки	2012-10-01	https://lkml.org/lkml/2012/9/1/94 commit	Исправлено в kernel 3.7-rc1
L0067	Утечка	staging: bcm: утечка ресурсов в bcm_init()	2012-10-01	https://lkml.org/lkml/2012/9/1/97 commit	Исправлено в kernel 3.7-rc1

Основные результаты работы

ИСП РАН

1. Разработан метод верификации драйверов устройств операционных систем для проверки выполнения правил корректного взаимодействия драйверов с ядром операционной системы;
2. Разработан метод построения моделей окружения драйверов устройств ОС Linux;
3. Разработан метод построения конфигурируемой системы верификации, обеспечивающий возможность расширения системы за счет пополнения набора правил корректности и набора инструментов верификации;
4. Разработаны методы оптимизации предикатной абстракции в инструменте BLAST;
5. На основе предложенных методов разработана система верификации драйверов ОС Linux.