Московский физико-технический институт
(государственный университет)
Факультет радиотехники и кибернетики
Кафедра информатики и вычислительной техники

Очистка остаточной информации в ОС “Эльбрус”

Выпускная квалификационная работа
(бакалаврская работа)

Студент: Паршин Дмитрий, гр. 313
Научный руководитель: Межуев Ю.В.

Москва, 2017

Постановка проблемы

Остаточная информация – данные, которые остаются на жестком диске и в оперативной памяти после их формального удаления операционной системой.

Причина возникновения остаточной информации:

для увеличения производительности многие операционные системы:

- без очистки содержимого удаляют индексный узел файла (inode)
- не очищают освобождаемую оперативную память

Возможно нежелательное восстановление данных из остаточной информации.

Цель

Реализация механизма очистки остаточной информации после удаления данных с жесткого диска и механизма очистки освобождаемых областей оперативной памяти при ее перераспределении на уровне ядра Linux путем перезаписи данных.

Задачи

- Выбор методов перезаписи
- Реализация выбранных методов
- Тестирование реализованного механизма

Требования

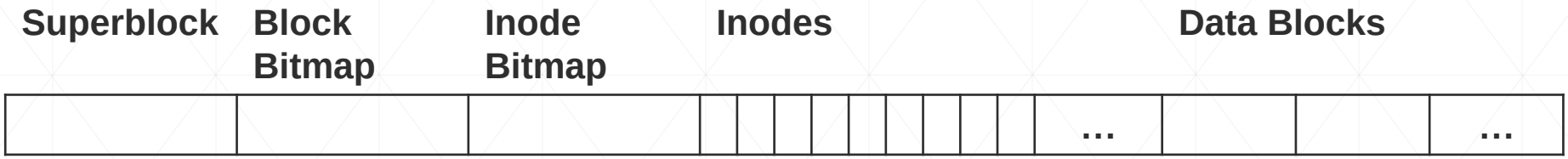
- Приемлемое время очистки
- Возможность выбора метода перезаписи
- Поддержка файловых систем ОС Linux – ext2, ext3, ext4

Методы перезаписи

Метод	Дата создания	Количество проходов	Шаблоны перезаписи			
			Все “0”	Все “1”	Случай-ные числа	Заданные последова-тельности
Российский стандарт “ГОСТ Р 50739-95”	1995	2	+		+	
Американский национальный стандарт Министерства обороны “DoD 5220.22-M”	2006	3			+	
Германский национальный стандарт “VSITR”	2009	7	+	+	+	
Метод Брюса Шнайера	1996	7	+	+	+	
Метод Питера Гутмана	1996	35			+	+

С ростом числа проходов улучшается качество удаления и время его выполнения. Исходя из простоты реализации и в соответствии с РД ФСТЭК «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от НСД к информации», были выбраны *метод Шнайера*, *перезапись всеми нулями* и *перезапись случайными числами*.

Структура файловой системы



Superblock – контейнер для высокоуровневых метаданных о файловой системе (суммарное число блоков, список свободных блоков и свободных индексных узлов (*inod'*ов), корневой индексный узел)

Block Bitmap – битовая карта свободных/занятых блоков

Inode Bitmap – битовая карта свободных/занятых индексных узлов

Inodes – таблица индексных узлов

Inode хранит в себе все метаданные для управления объектами файловой системы (время создания, id пользователя, права доступа, возможные файловые операции).

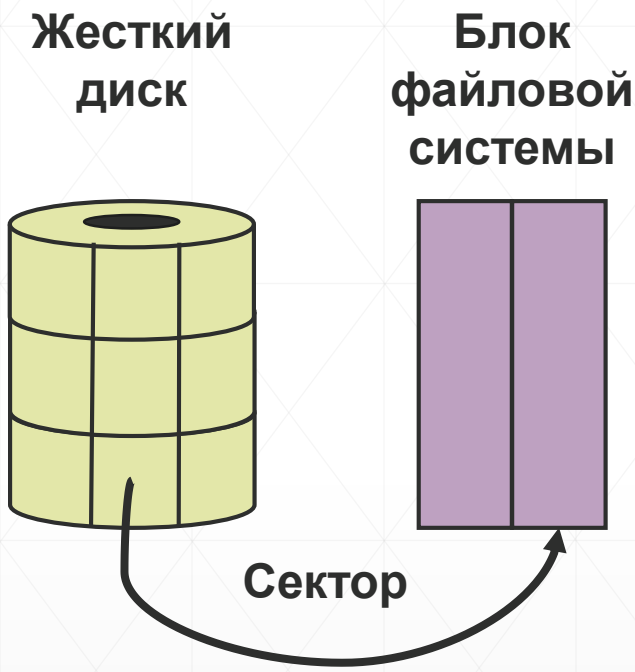
Data Blocks – блоки полезных данных, т.е. содержимое файлов и директорий

Соотношение систем адресации жесткого диска и файловой системы

Наименьший адресуемый элемент жесткого диска называется сектором, размером 512 байт. Минимально адресуемая единица файловой системы – блок, состоящий их одного или нескольких секторов и расположенный в оперативной памяти.

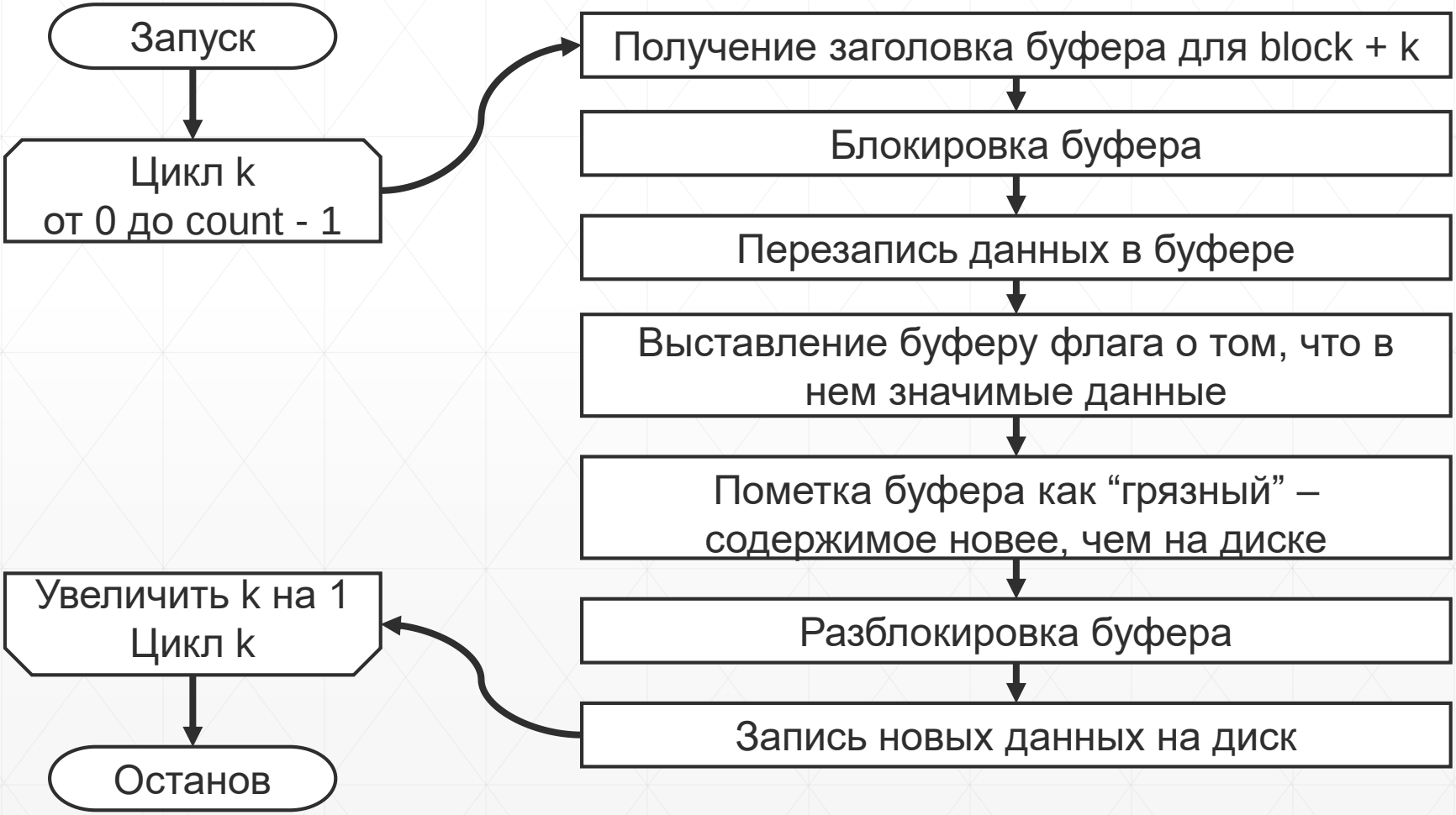
Изменения данных происходят в блоках, для синхронизации данных в блоках с данными на жестком диске вызываются специальные функции записи на диск.

Блок хранится в структуре данных, называемой *буфером* (*buffer*), которая описывается своим дескриптором, *заголовком буфера* (*buffer head*).



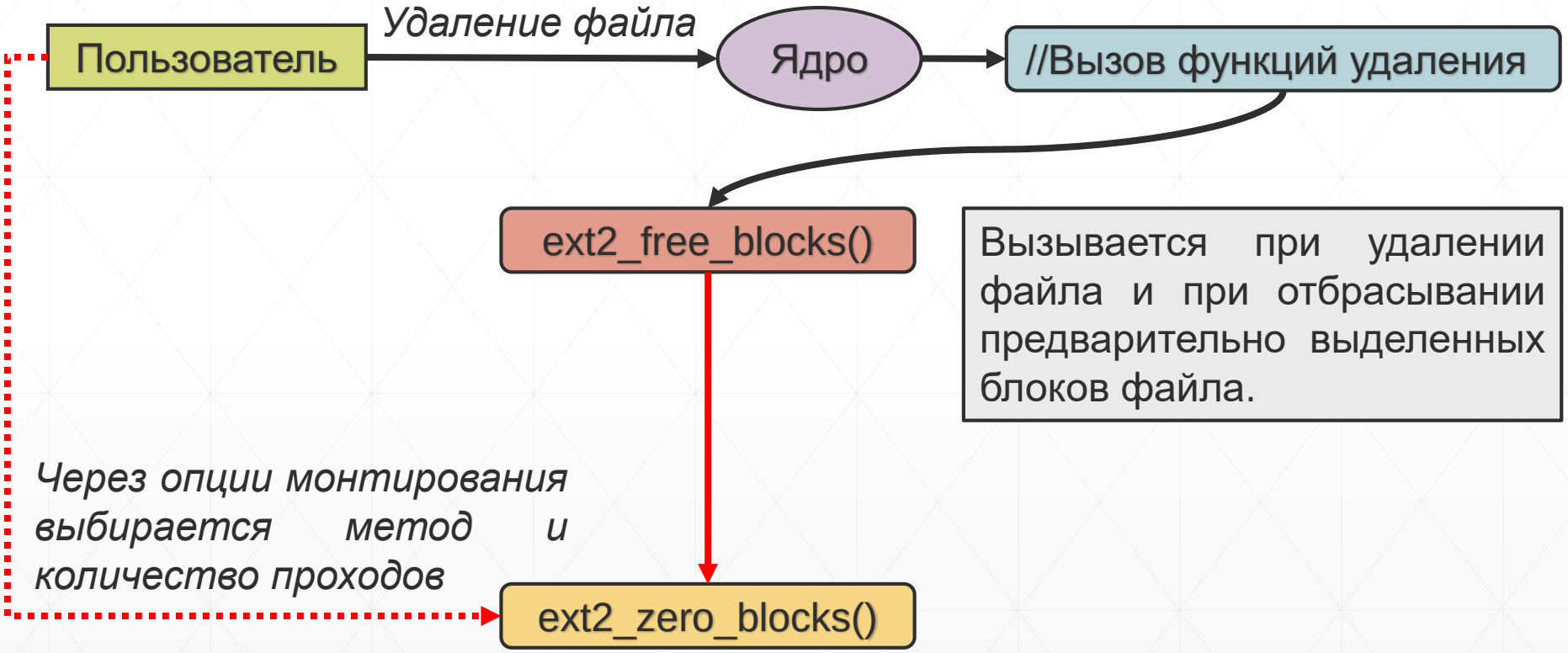
Реализованный алгоритм очистки для жесткого диска

Функция перезаписи *ext2_zero_blocks*, принимающая на вход суперблок, стартовый блок *block* и количество блоков *count*, выбирает метод перезаписи и для каждого прохода перезаписи выполняет следующее:



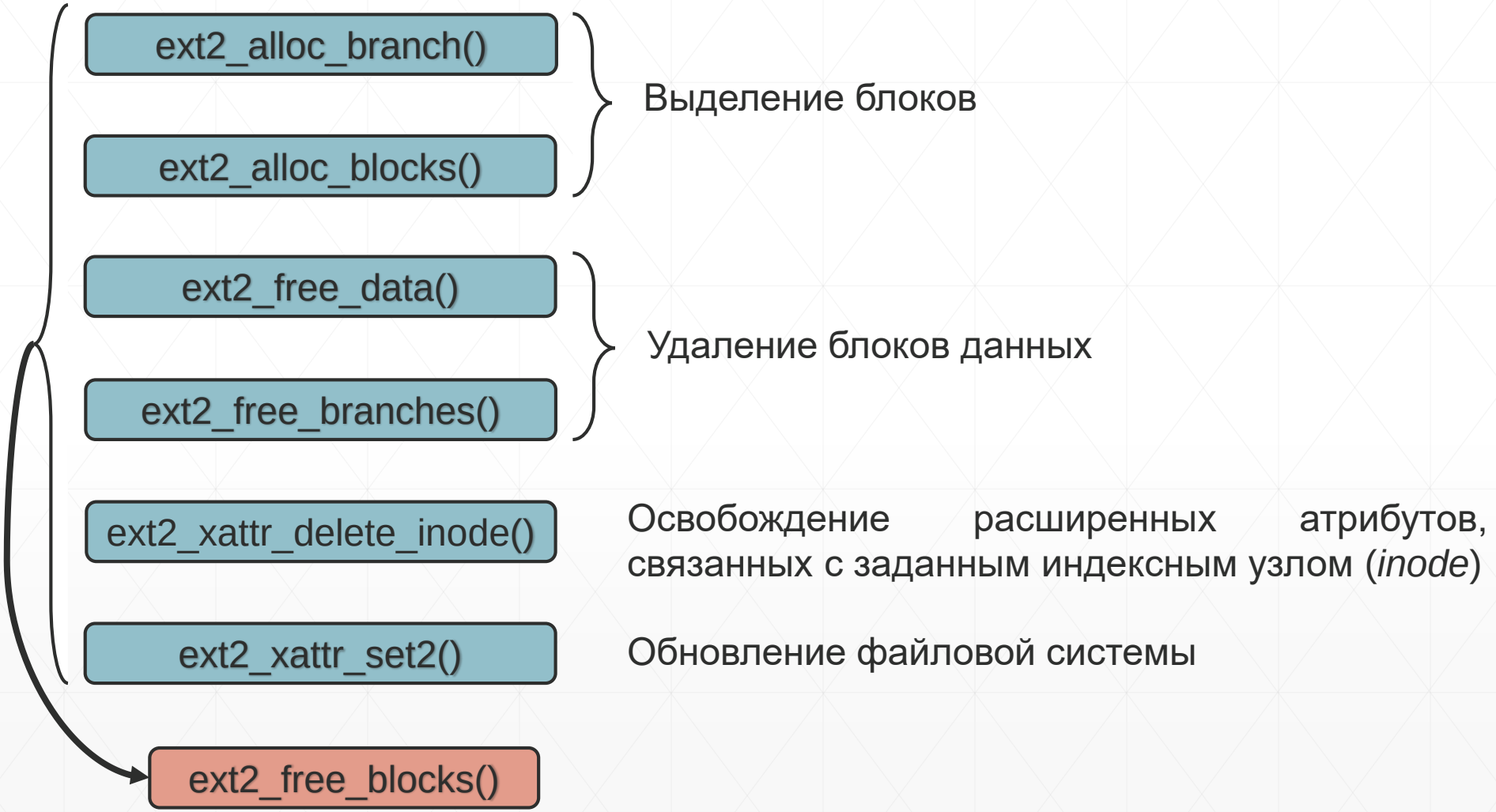
Очистка остаточной информации в ext2/ext3

Реализованный механизм очистки *secdel*

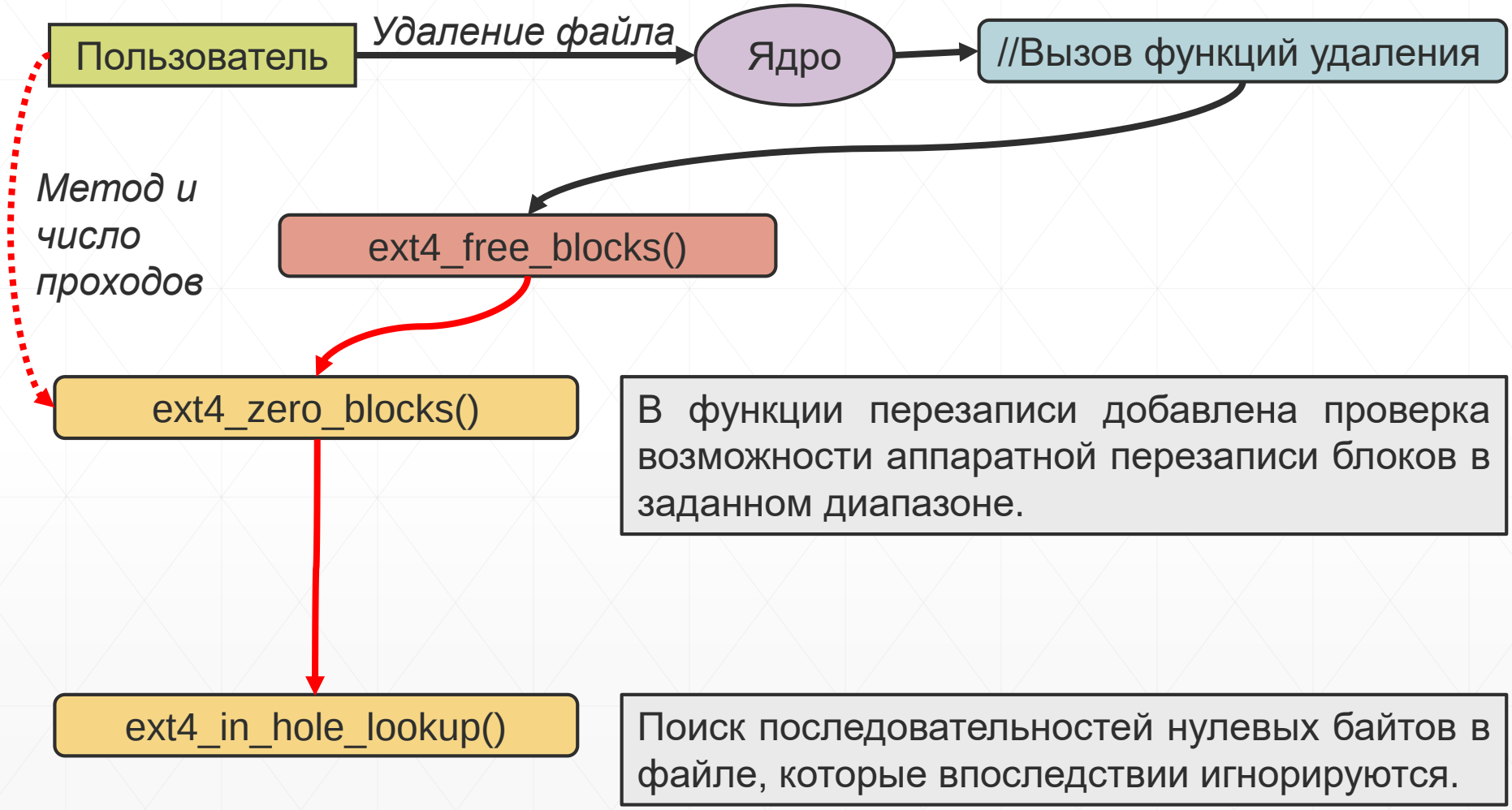


В ext3 реализация очистки остаточной информации аналогична ext2.

Функции удаления данных в ext2



Очистка остаточной информации в ext4



В функции перезаписи добавлена проверка возможности аппаратной перезаписи блоков в заданном диапазоне.

Поиск последовательностей нулевых байтов в файле, которые впоследствии игнорируются.

Работа с оперативной памятью в ядре Linux

Оперативная память организована в виде *страниц* размером 4 КБ и управляется распределителем памяти *buddy.* Когда требуется память меньшего размера, используются распределители *slab*, *slub*, *slob*.

- **Buddy Allocator**

Делит память на разделы с размером, кратным степени 2, и пытается распределить запросы памяти, используя алгоритм *best-fit*. Когда пользователь освобождает память, то свободные смежные блоки объединяются для минимизации фрагментирования.

- **Slab**

Управляет программными кэшами ядра ОС, в которых размещены часто используемые объекты данных определенного типа и размера.

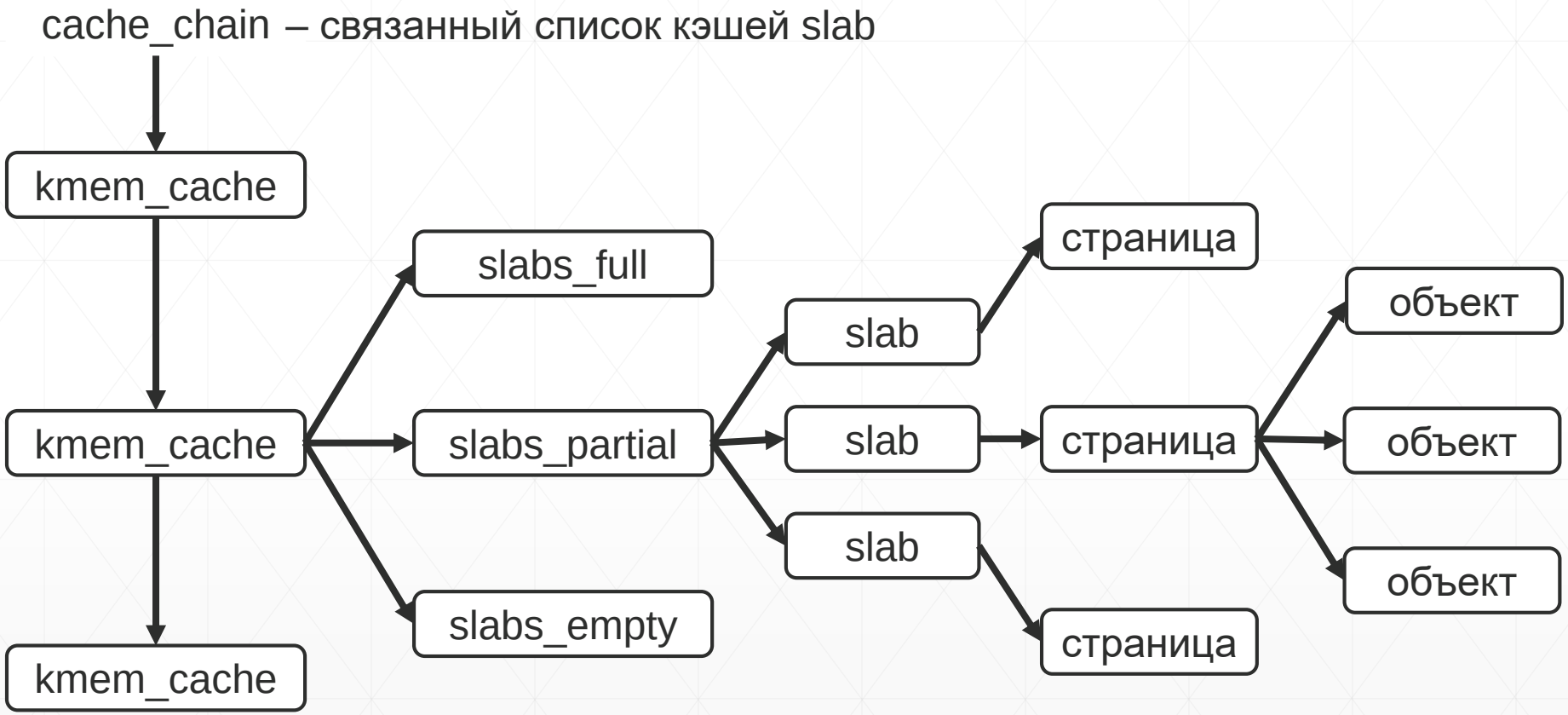
- **Slob**

Применяется для небольших и встроенных систем.

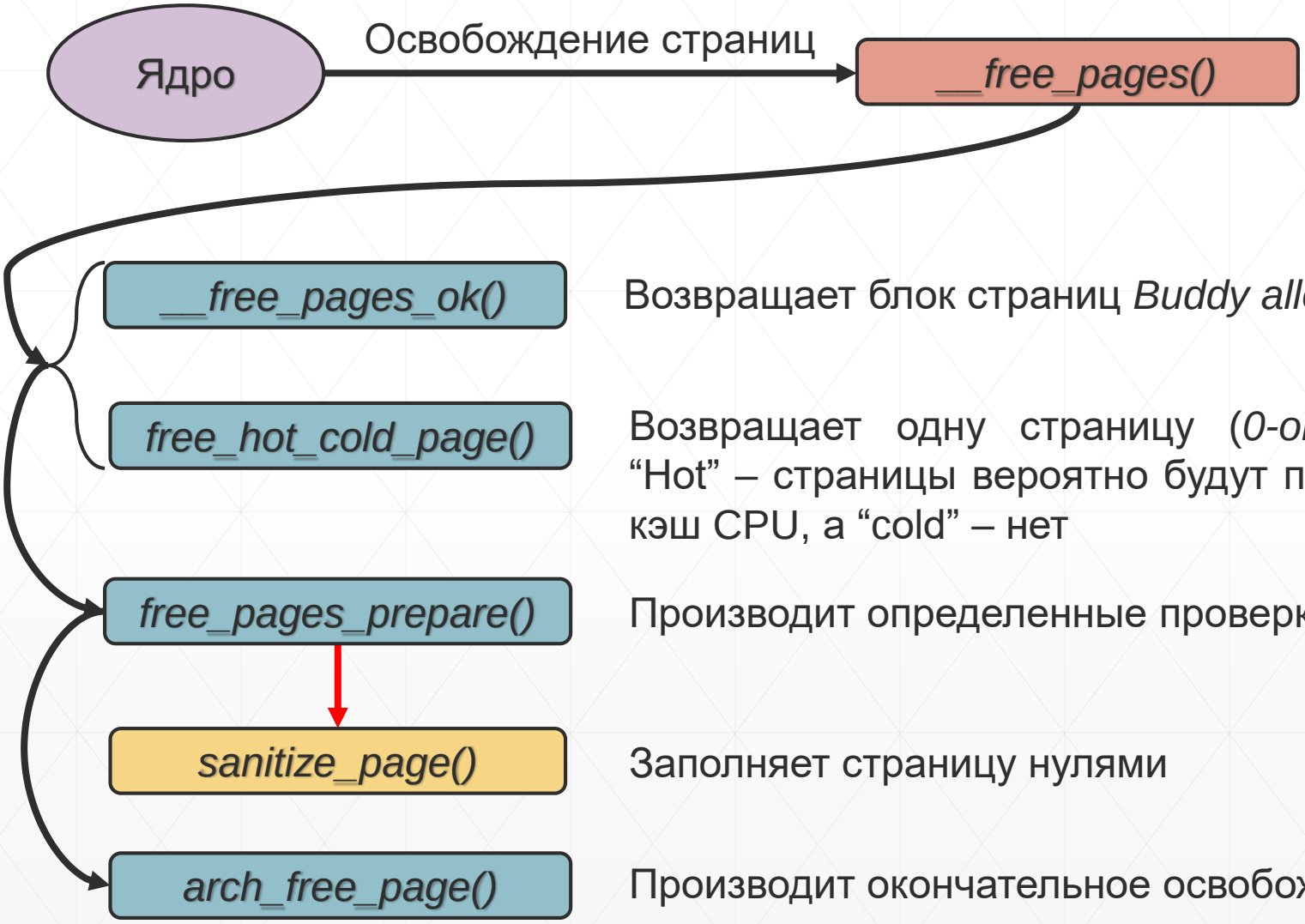
- **Slub**

Предназначен для систем с большими объемами RAM.

Схема распределителя памяти SLAB



Очистка страниц



Возвращает блок страниц *Buddy allocator*'у

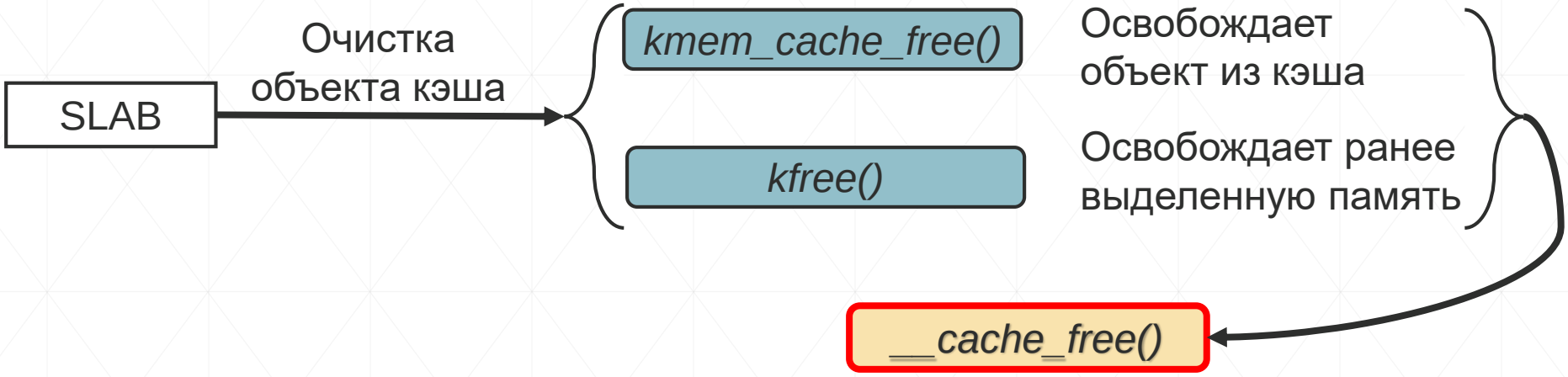
Возвращает одну страницу (*0-order page*). "Hot" – страницы вероятно будут помещены в кэш CPU, а "cold" – нет

Производит определенные проверки

Заполняет страницу нулями

Производит окончательное освобождение страниц

Очистка закэшированных объектов



В функционал *__cache_free()* добавлена перезапись объектов нулями. Аналогично модифицированы *slob_free()* и *slab_free()* для распределителей памяти SLOB и SLUB.

Тестирование очистки остаточной информации на жестком диске

Тестирование было проведено с помощью теста *secdel_tc* из системы OSLTC.

Алгоритм тестирования:

1) Создается тестовая файловая система (ТФС).
2) ТФС монтируется без опций очистки.
3) В ТФС создается файл с тестовой сигнатурой.
4) Файл удаляется.
5) ТФС отмонтируется.
6) Проверяется наличие сигнатуры в ТФС утилитой *fgrep*.
7) ТФС монтируется с опциями очистки.
8) Создается объект с тестовой сигнатурой.
9) Объект удаляется.
10) ТФС отмонтируется.
11) Производится повторный поиск сигнатуры.

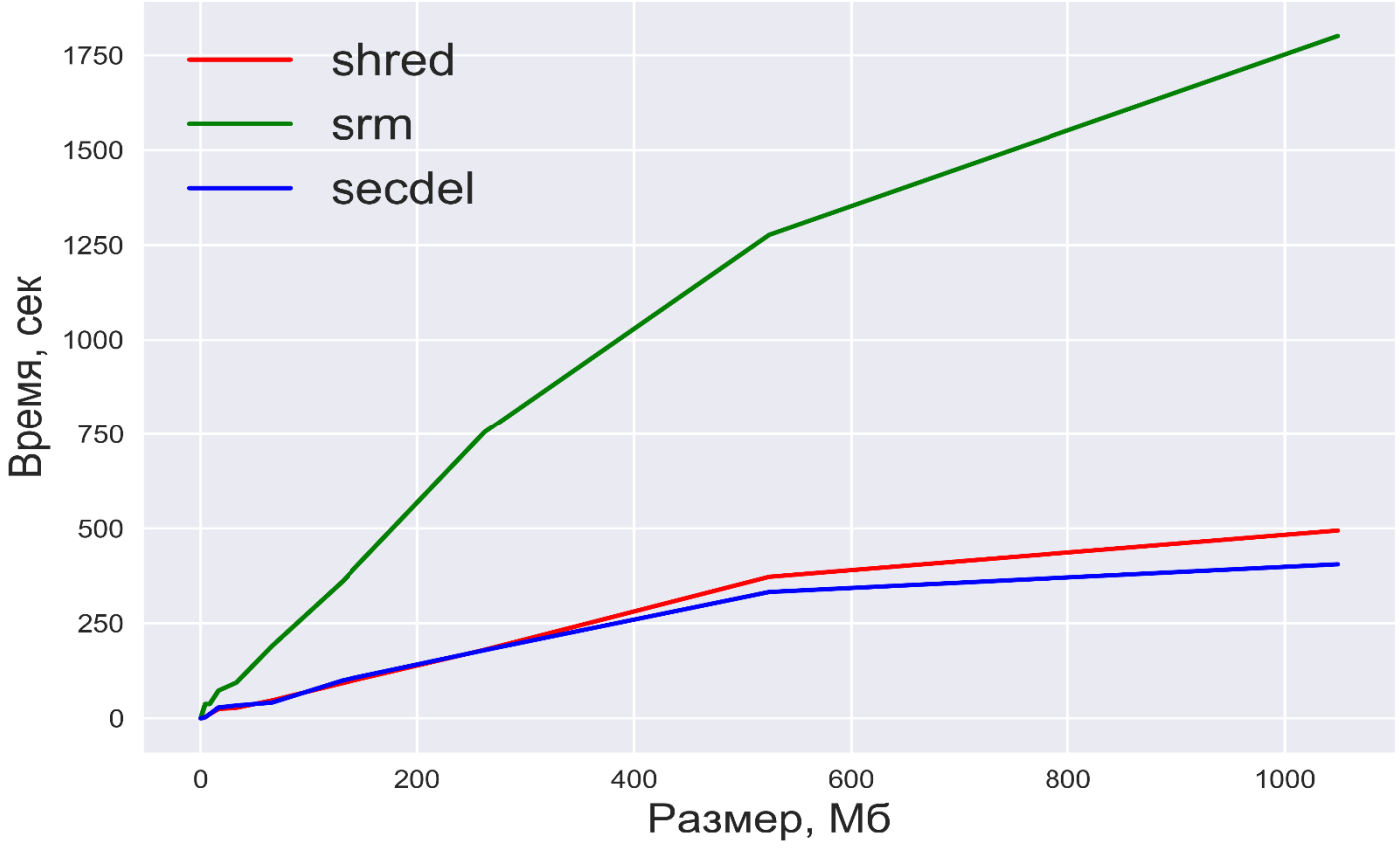
Если не найдена, то тест считается пройденным (PASS).

Тестирование очистки остаточной информации на жестком диске. Пример

```
bash-4.2# ./run
=== Create /home/tester/OSLTC/tests/trust/secdel_tc/results-4/scen
=== Пн июн 26 15:50:58 MSK 2017
=== Linux trust-pc2 2.6.33-elbrus.033.16.95 #1 SMP Mon May 1 02:44:37 UTC 2017 e
2k E2S MONOCUB GNU/Linux
=== cwd: /home/tester/OSLTC/tests/trust/secdel_tc
=== Start:  15.50.58: .: ./bin/secdel_tc  -u srm
=== attr:   root
=== ./bin/secdel_tc  -u srm
=== Finish: 15.51.01: .: ./bin/secdel_tc  -u srm: PASS
=== Start:  15.51.02: .: ./bin/secdel_tc  -u shred
=== attr:   root
=== ./bin/secdel_tc  -u shred
=== Finish: 15.51.05: .: ./bin/secdel_tc  -u shred: PASS
=== Start:  15.51.05: .: ./bin/secdel_tc  -u kernel_del
=== attr:   root
=== ./bin/secdel_tc  -u kernel_del
=== Finish: 15.51.08: .: ./bin/secdel_tc  -u kernel_del: PASS
=== PASS
===
=== TOTAL LINES: 3
=== PASS:      3
=== FAIL:      0
bash-4.2#
```

Тестирование очистки остаточной информации на жестком диске. Производительность

Для сравнения реализованного механизма *secdel* с существующими утилитами *shred* и *srm* были использованы 35 проходов перезаписи случайными числами:



Быстрее, чем *shred* примерно на 9%

Тестирование очистки остаточной информации в оперативной памяти

Для тестирования очистки оперативной памяти была проведена атака методом “холодной” перезагрузки (*cold boot attack*).

Метод основан на способности оперативной памяти какое-то время сохранять информацию после отключения питания. Это дает возможность взломщику прочитать ее содержимое просто выключив и включив компьютер и затем запустив собственную операционную систему.

Алгоритм тестирования:

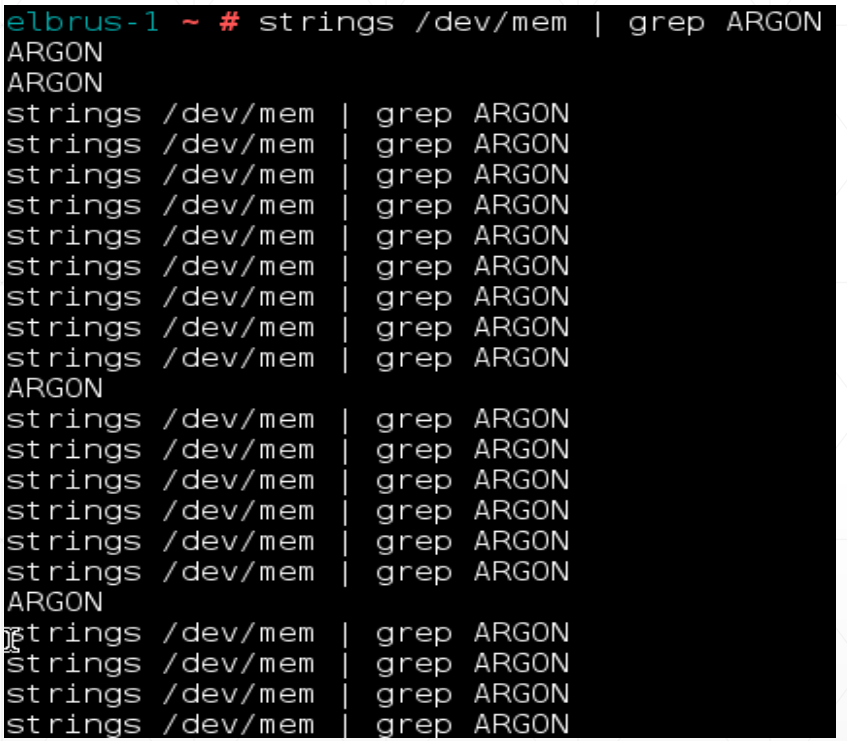
1) Запуск *sync* для сброса любых кэшированных данных на диск.
2) Запуск программы, написанной на Python, которая заполняет память словом "ARGON".
3) Аппаратный перезапуск системы по истечении нескольких минут работы Python программы.
4) Поиск слова "ARGON" в памяти.

Если слово найдено, то какая-то часть памяти выжила при перезагрузке. Если не обнаружено, то очистка оперативной памяти работает.

Тестирование очистки остаточной информации в оперативной памяти. Результаты

Очистка включена

Очистка отключена



Использование перезаписи оперативной памяти оказывает влияние на производительность системы. Среднее время компиляции ядра в однопоточном режиме увеличивается примерно на 3%. Без очистки – 648 секунд, с очисткой – 664 секунды.

Результаты работы

- Реализован и протестирован механизм очистки остаточной информации для жесткого диска и оперативной памяти
- Программная реализация очистки остаточной информации передана для внедрения в штатную версию ядра ОС “Эльбрус” на базе Linux 3.14
- Доработан тест механизма очистки. Тест передан для включения в систему тестирования OSLTC из состава ОПО “Эльбрус”