Итак, вы решили надежно записывать данные на диск

Дмитрий Родионов

Picodata

О себе

- Занимаюсь базами данных и распределенными системами
- Ведущий разработчик в Picodata
- Аспирант в ИСП РАН

Почему о надежности записи?

- Как любой разработчик, захотел написать велосипед

Почему о надежности записи?

- Как любой разработчик, захотел написать велосипед
- БД начинается со стораджа

Почему о надежности записи?

- Как любой разработчик, захотел написать велосипед
- БД начинается со стораджа
- Написал!

Почему о надежности записи?

- Как любой разработчик, захотел написать велосипед
- БД начинается со стораджа
- Написал!

Как верить тому, что написал?

О чем доклад:

- Надежно записывать данные — сложнее, чем кажется

О чем доклад:

- Надежно записывать данные — сложнее, чем кажется
- На каких ошибках в популярном софте мы можем научиться

О чем доклад:

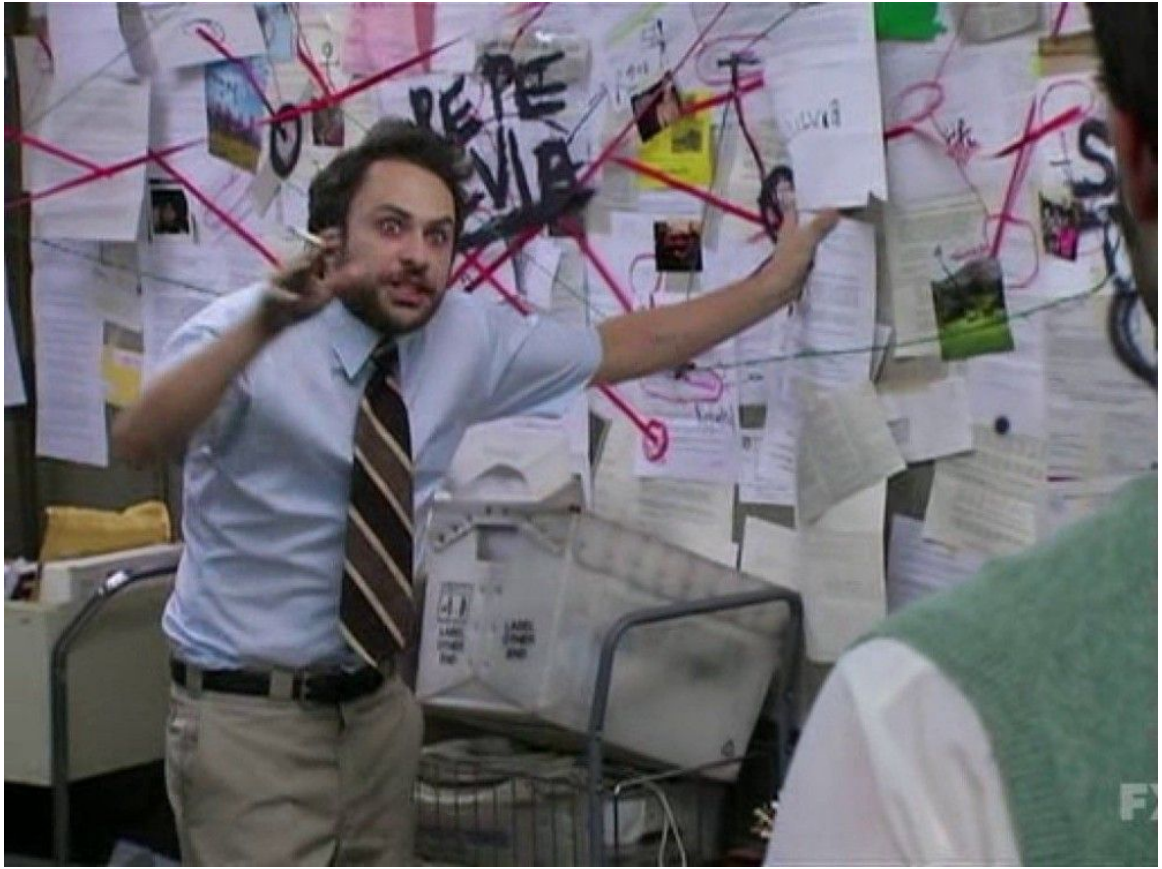
- Надежно записывать данные — сложнее, чем кажется
- На каких ошибках в популярном софте мы можем научиться
- Сила failure injection

О чем доклад:

- Надежно записывать данные — сложнее, чем кажется
- На каких ошибках в популярном софте мы можем научиться
- Сила failure injection
- Перспективы применения формальных методов

О чем доклад:

- Надежно записывать данные — сложнее, чем кажется
- На каких ошибках в популярном софте мы можем научиться
- Сила failure injection
- Перспективы применения формальных методов
- Что это значит для велосипедостроения



Будет много ссылок + библиография в конце

О чем доклад:

О проблемах, с которыми приходится сталкиваться при обеспечении долговечности в системах хранения данных

Об инструментах и подходах, существующих и перспективных, призванных ~~обуздать хаос~~ справиться с имеющимися вызовами

Что сложного? Берем да записываем

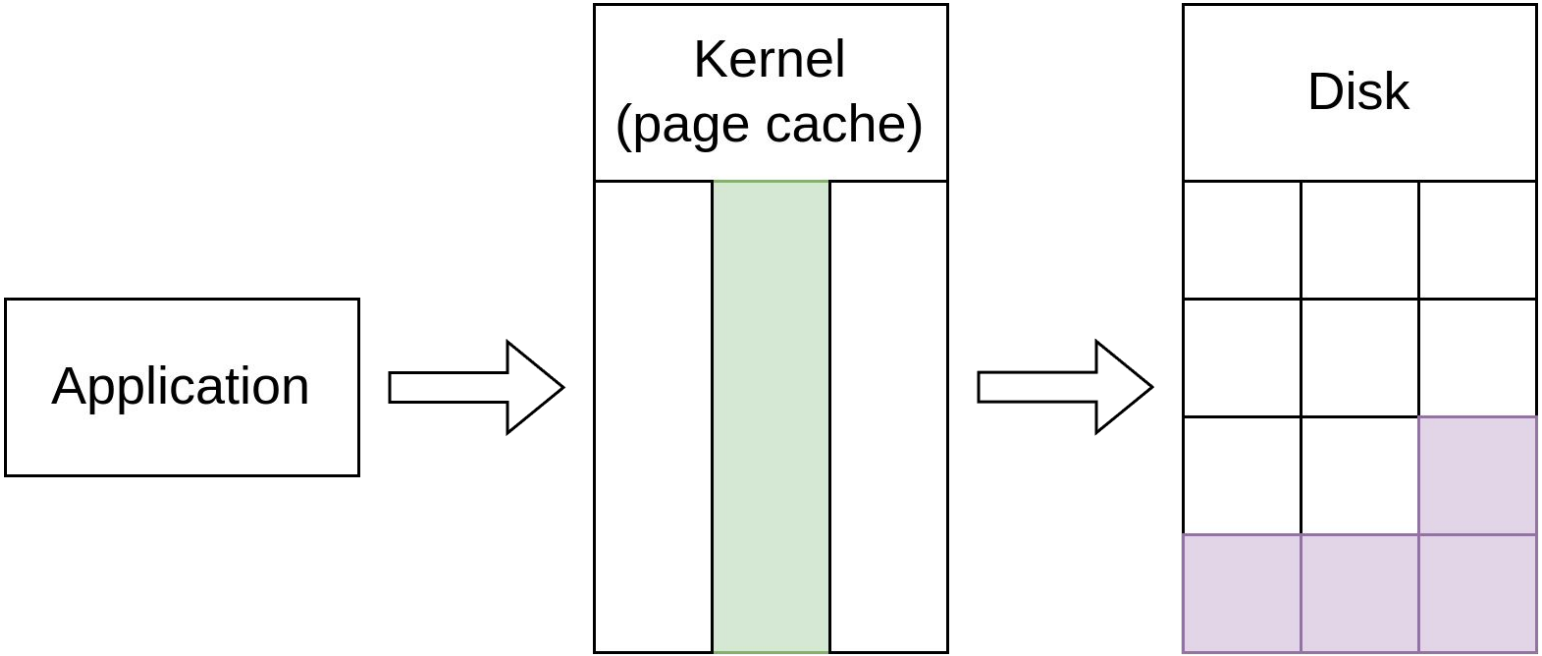
```
with open("hello", "w+") as f:
    f.write("hey")
```

Ввод/вывод для блочных устройств

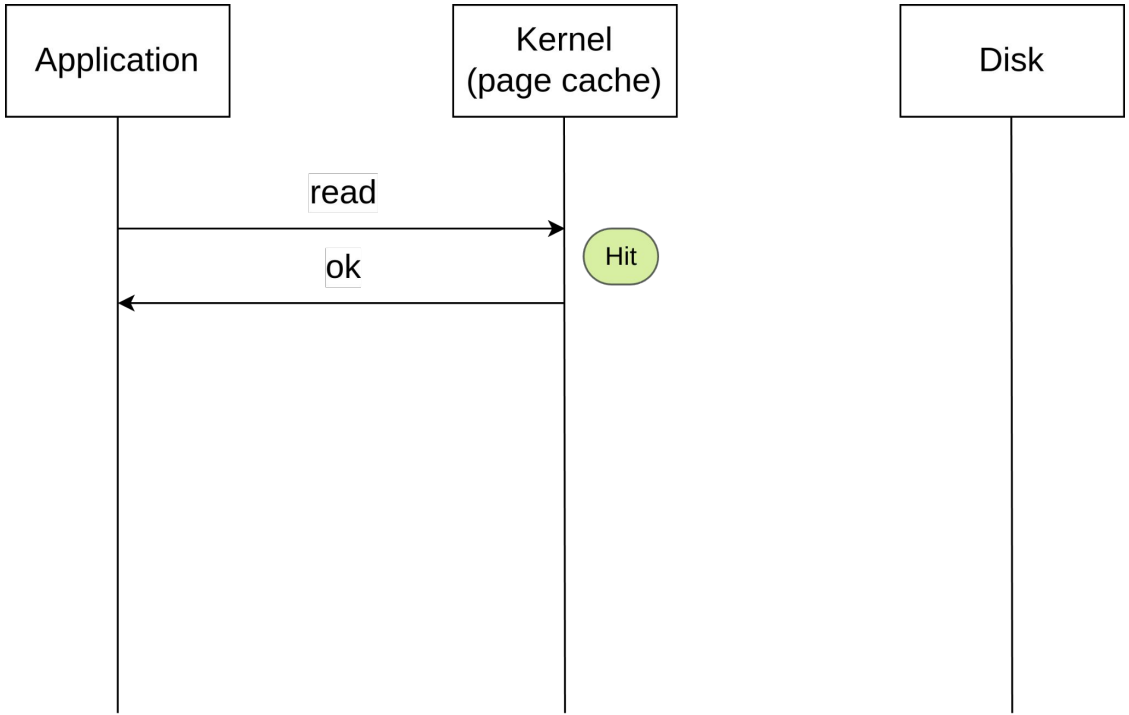
Блочный ввод/вывод

- Диски — блочные устройства
- Внезапно: блок пишем, блок читаем
- Диск медленный, амортизируем задержку

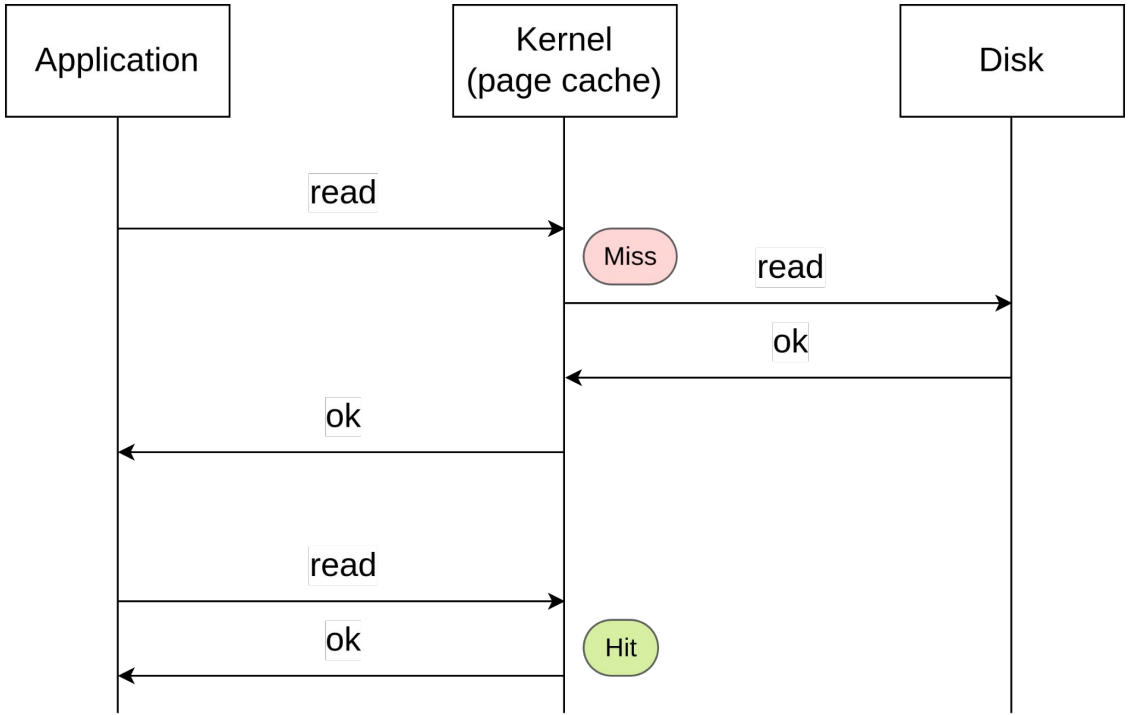
Page cache



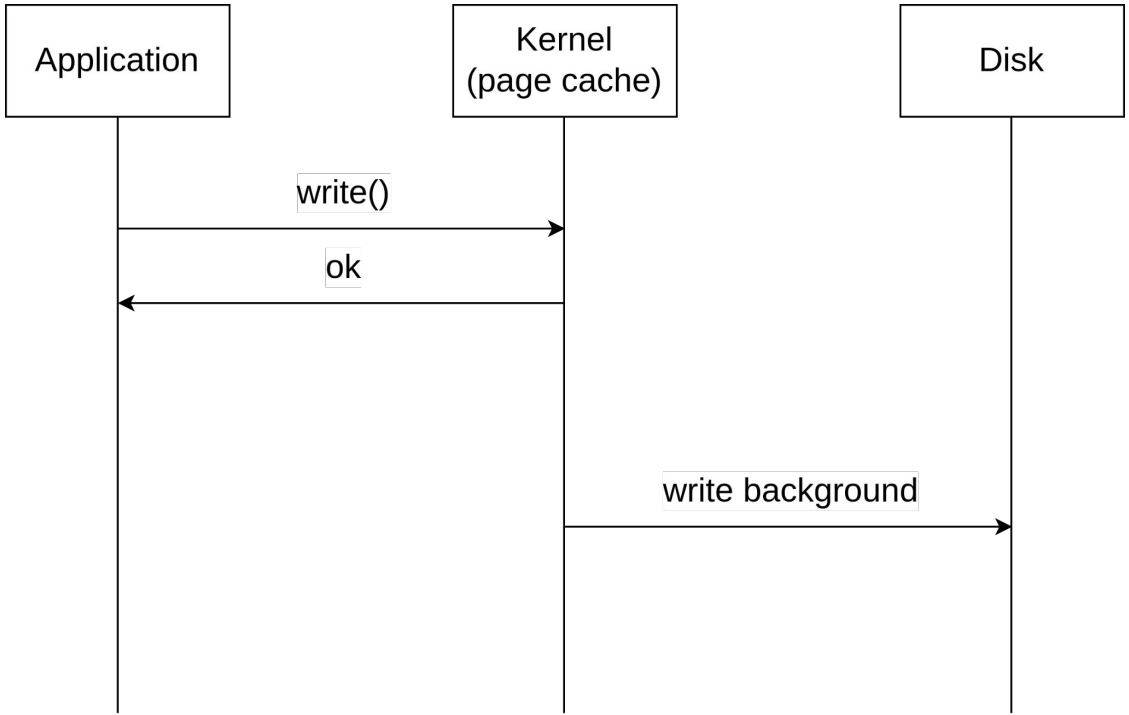
Read: cache hit



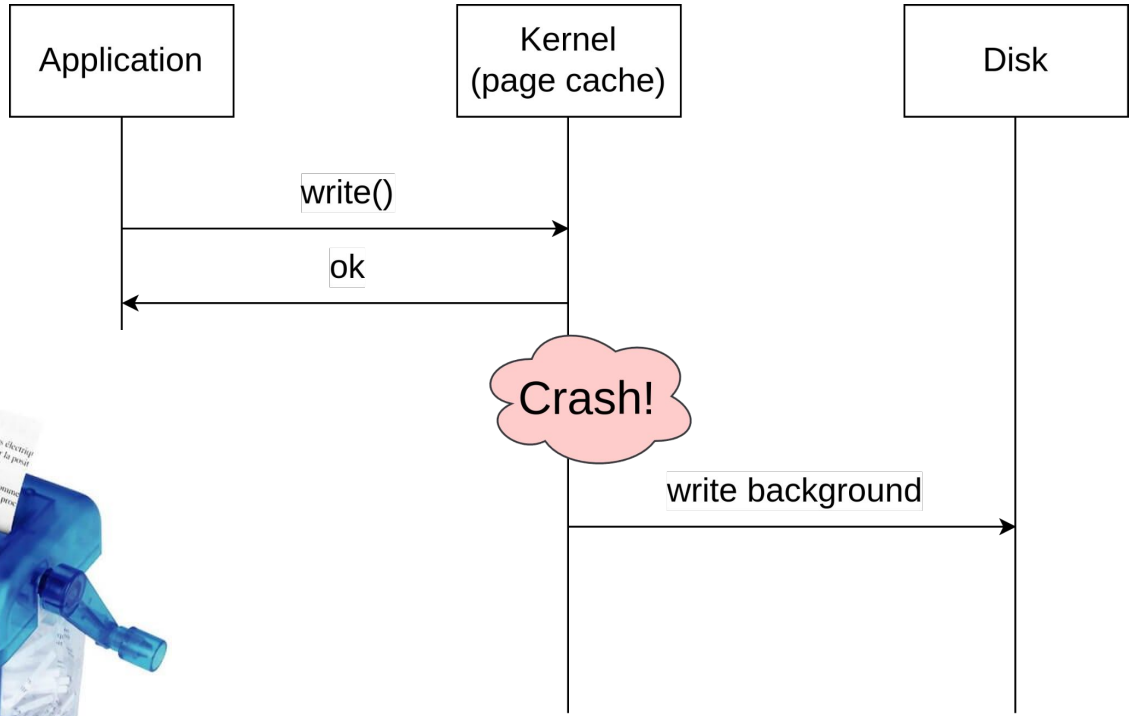
Read: cache miss



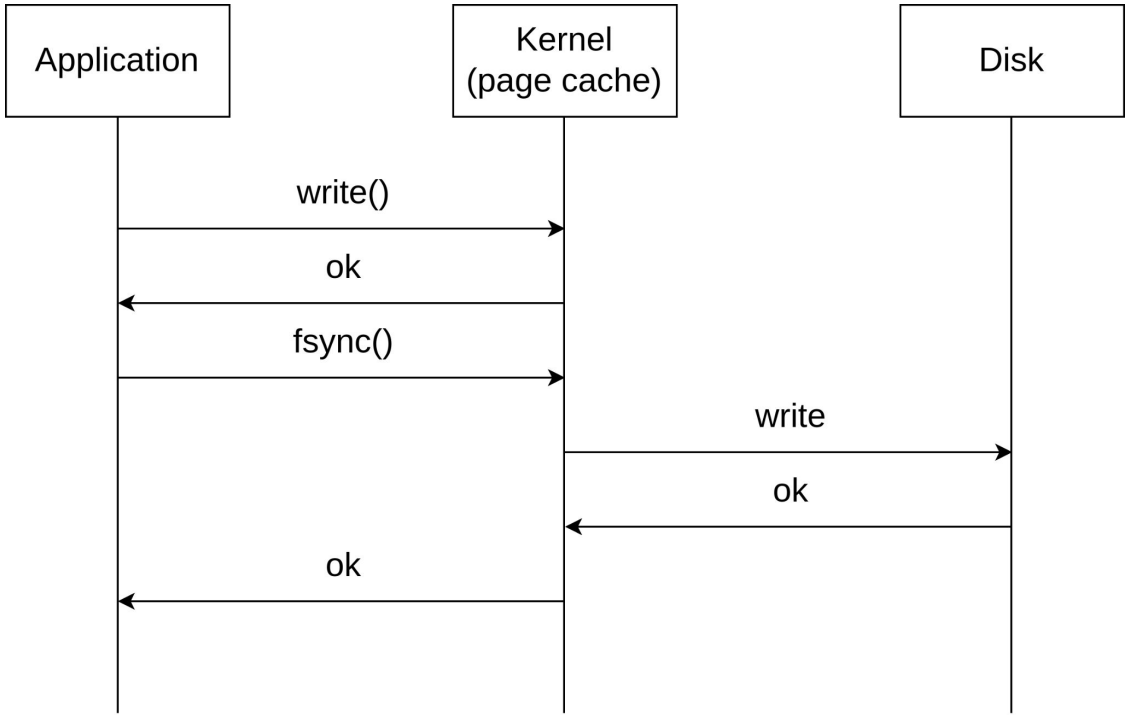
Write: async flush



Write: async flush — crash!



Write: fsync



Разные способы выполнить flush

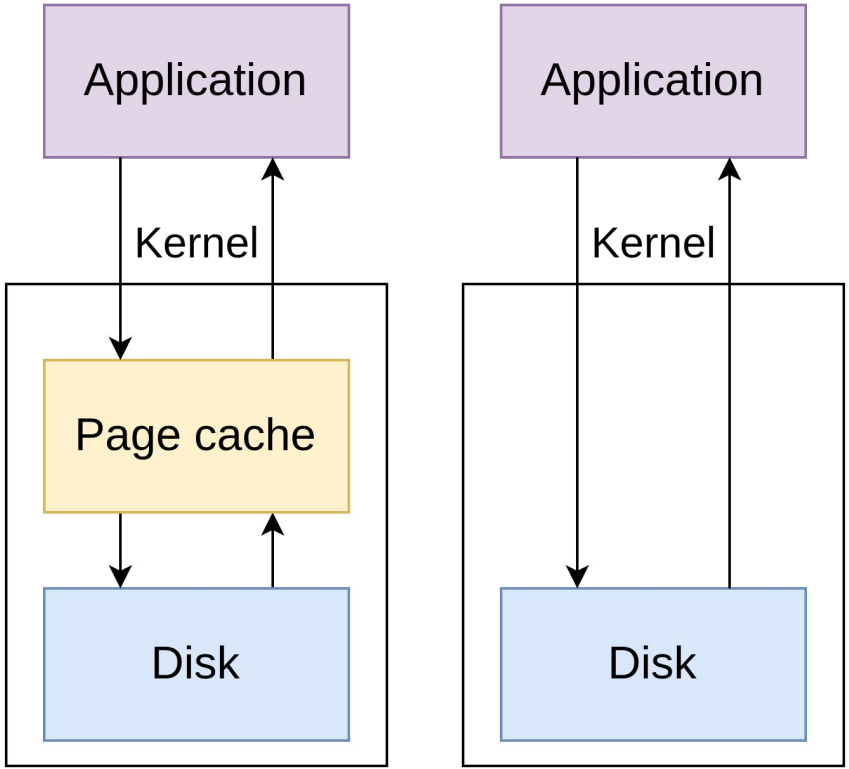
- fsync
- fdatasync
  - отличается от fsync, отличается на Linux и FreeBSD
- sync_file_range (Linux only)
- msync (при работе с файлами через mmap)

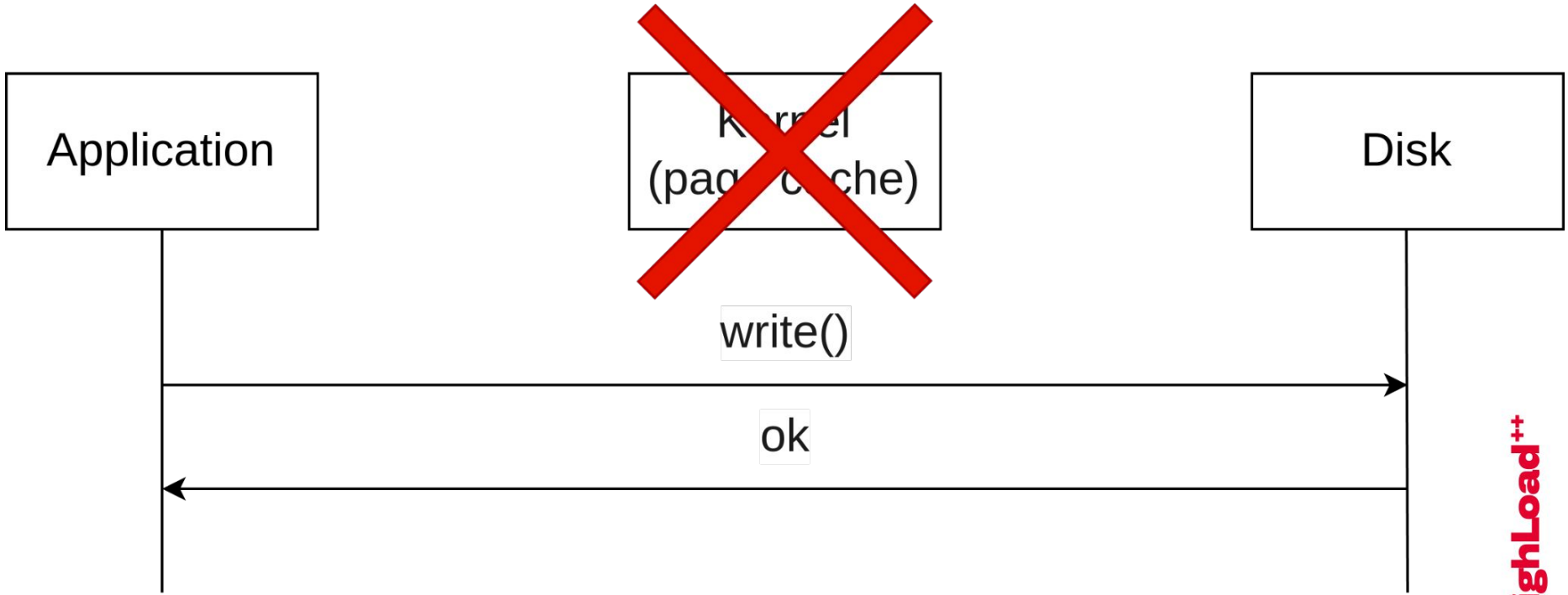
Можно без кеша как-то?

- O_DIRECT (Direct IO)

Может быть, можно без кеша как-то?

Direct IO




Application
Disk
write()
ok

Что за зверь O_DIRECT

- Противоречивая штука

Clarifying Direct IO's Semantics

Что за зверь O_DIRECT

- Противоречивая штука
- Долгое время без спецификации

Clarifying Direct IO's Semantics

Немного истории O_DIRECT

The exact meaning of O_DIRECT has historically been negotiated in non-public discussions between powerful enterprise database companies and proprietary Unix systems, and its behaviour has generally been passed down as oral lore rather than as a formal set of requirements and specifications

Clarifying Direct IO's Semantics

Немного истории O_DIRECT

"The thing that has always disturbed me about O_DIRECT is that the whole interface is just **stupid**, and was probably designed by a **deranged monkey** on some serious mind-controlling substances."

Linus

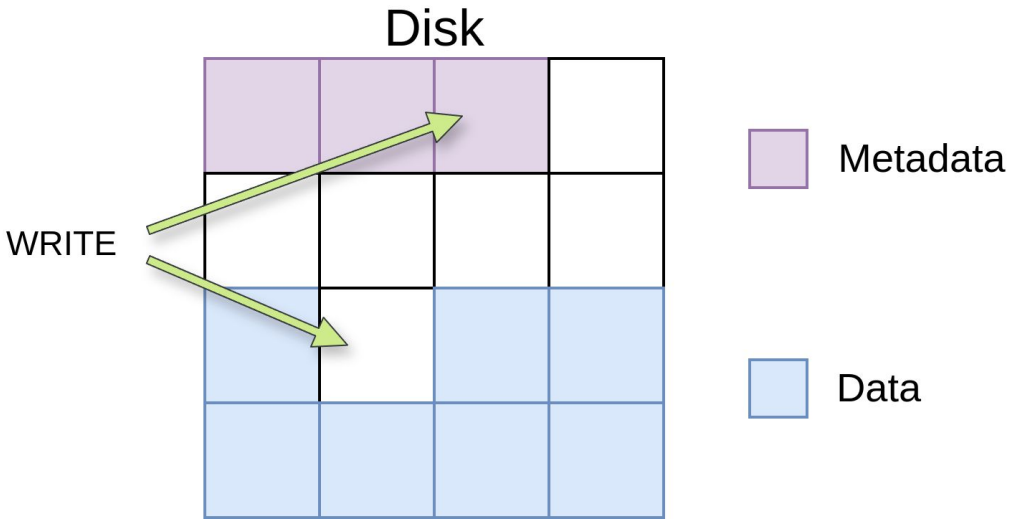
Clarifying Direct IO's Semantics

O_DIRECT != fsync

- Диск имеет собственный write cache

O_DIRECT != fsync

- Диск имеет собственный write cache
- Особенности записи метаданных при выделении новых блоков



Ensuring data reaches disk

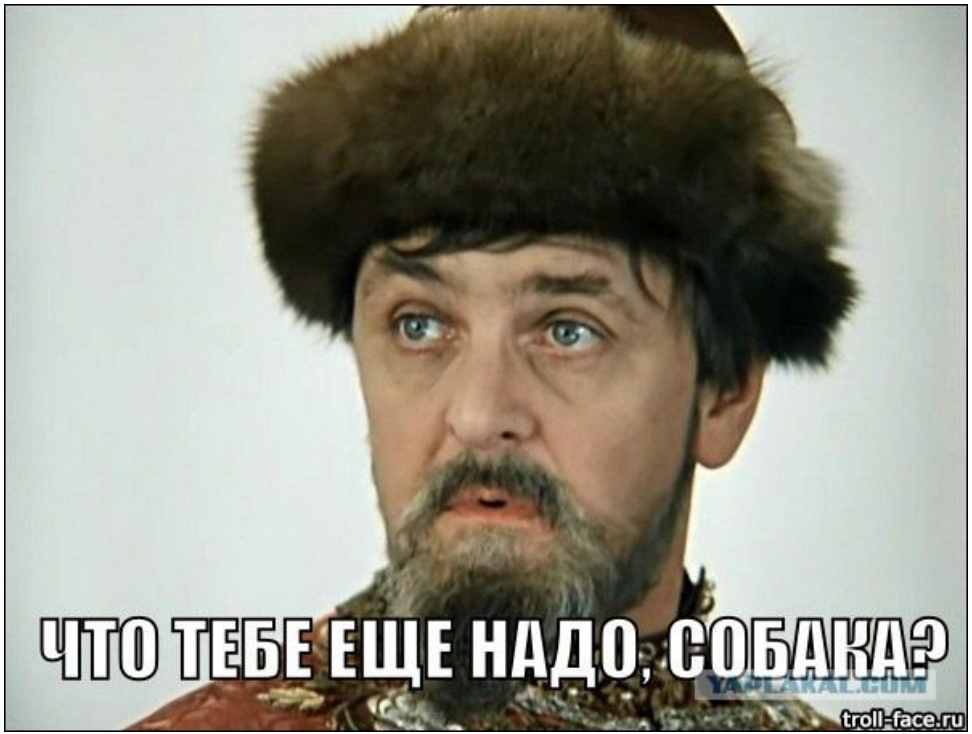
Clarifying Direct IO's Semantics (§Allocating writes)

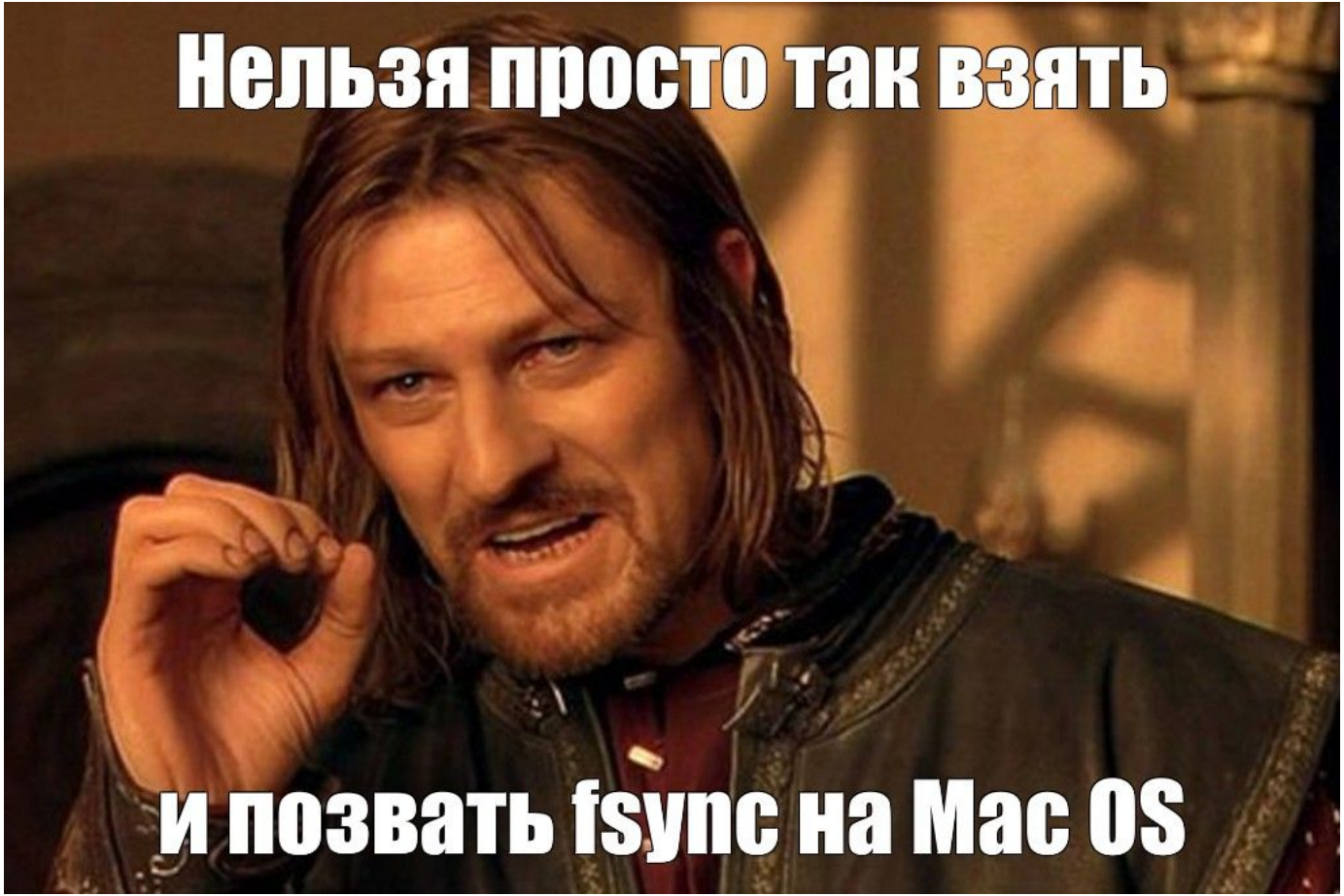
Ну, теперь-то все хорошо?

Игнорирование FLUSH

Coerced cache eviction and discreet mode journaling: Dealing with misbehaving disks

Производитель 👍 fsync — ok





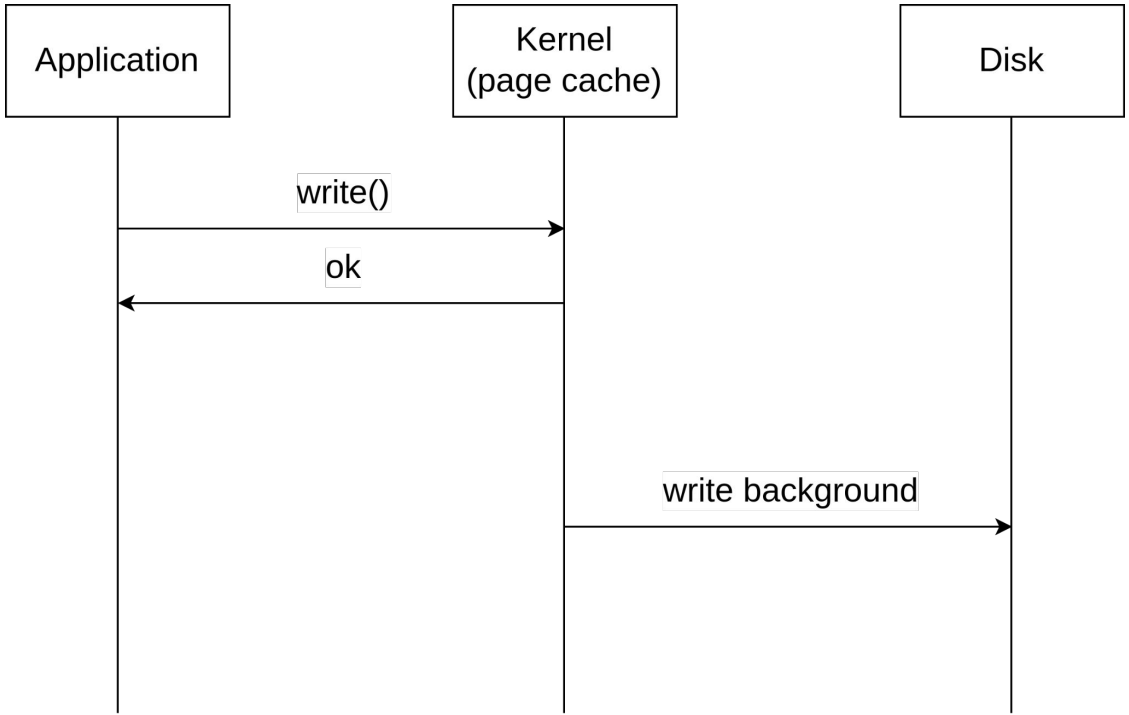
F_FULLFSYNC

Note that while fsync() will flush all data from the host to the drive the drive itself may not physically write the data to the platters for quite some time and it may be written in an out-of-order sequence.

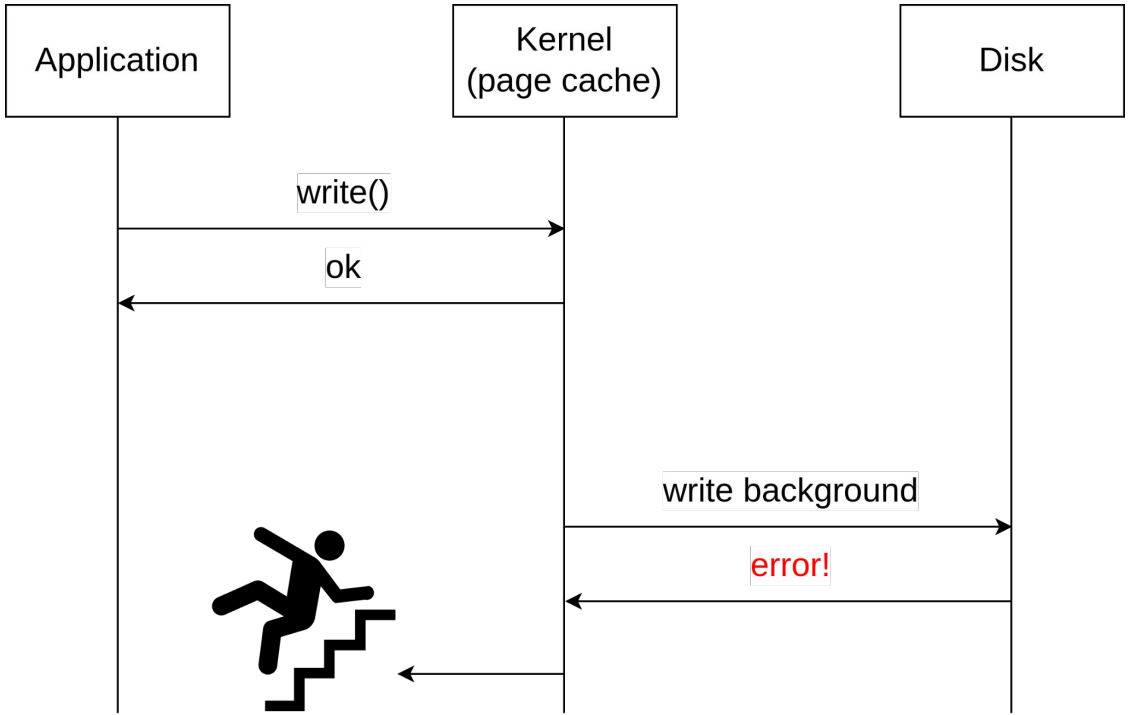
fcntl(F_FULLFSYNC)

man 2 fsync

Допущение фоновой записи

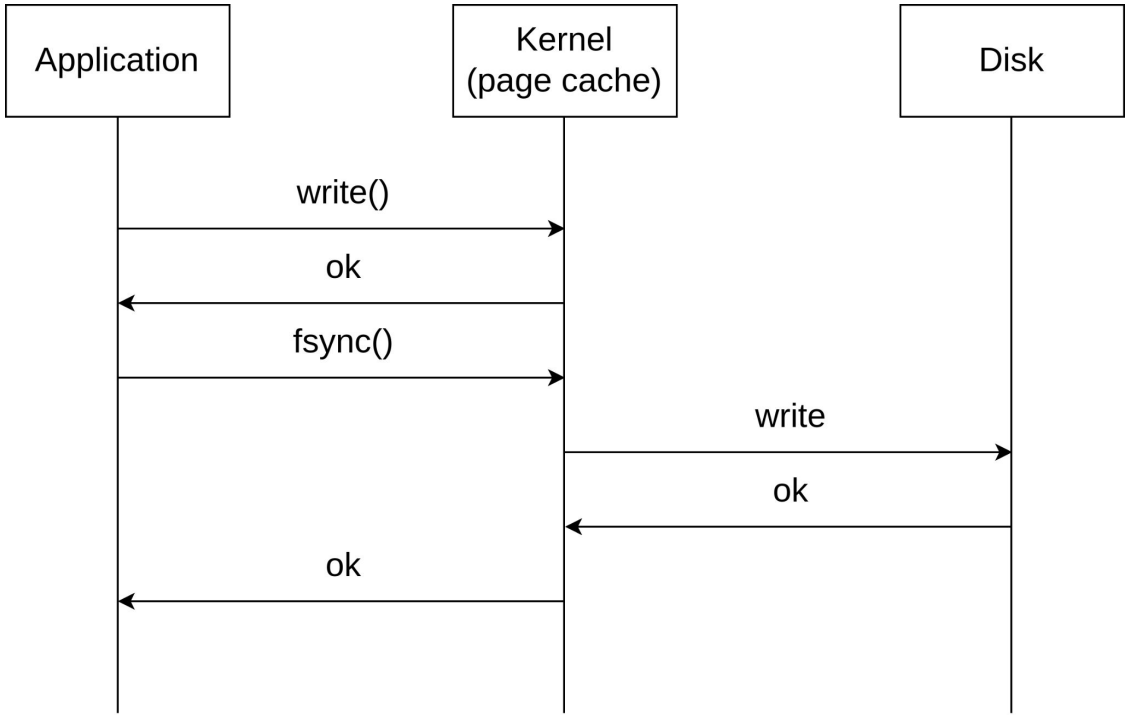


Ошибка при фоновой записи



man 2 close

fsync вылавливает ошибки



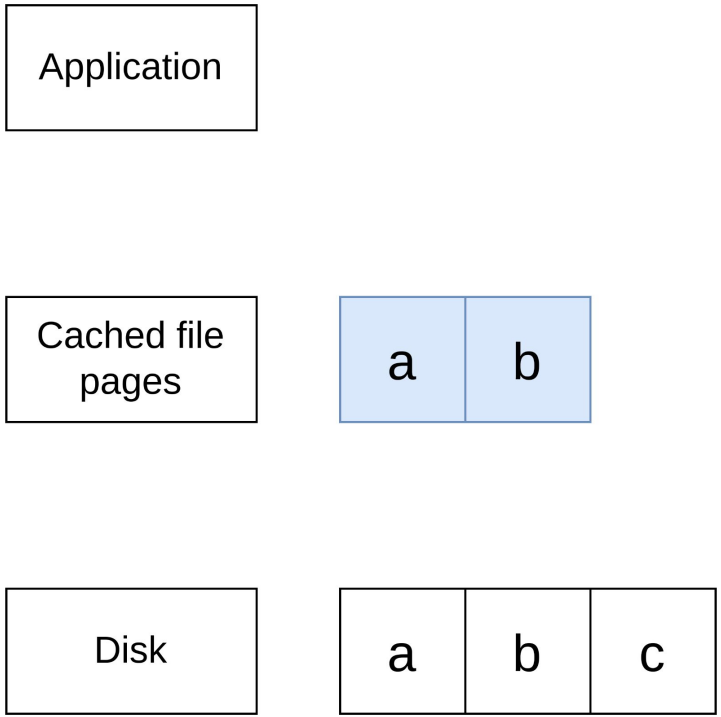
fsyncgate

PostgreSQL's handling of fsync() errors is unsafe and risks data loss

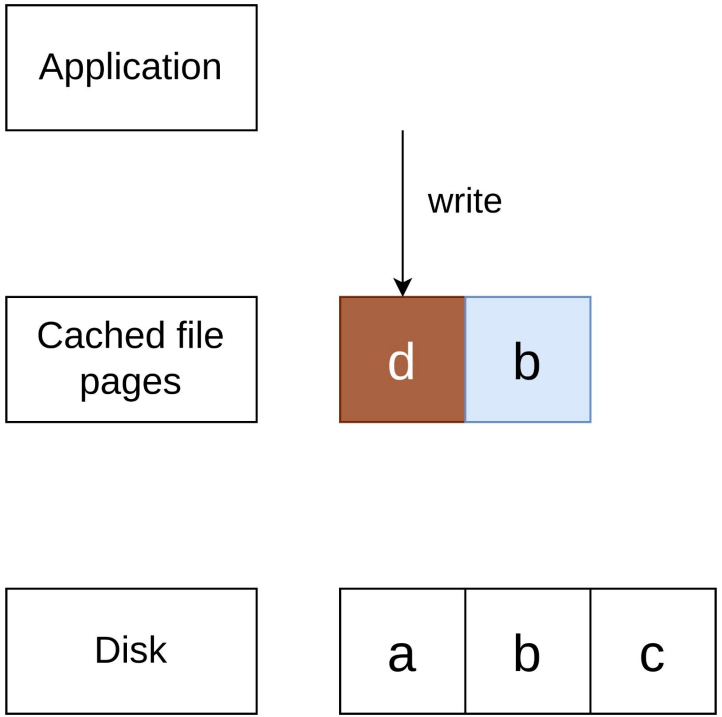
Lists:pgsql-hackers

[PostgreSQL's handling of fsync() errors is unsafe and risks data loss at least on XFS]

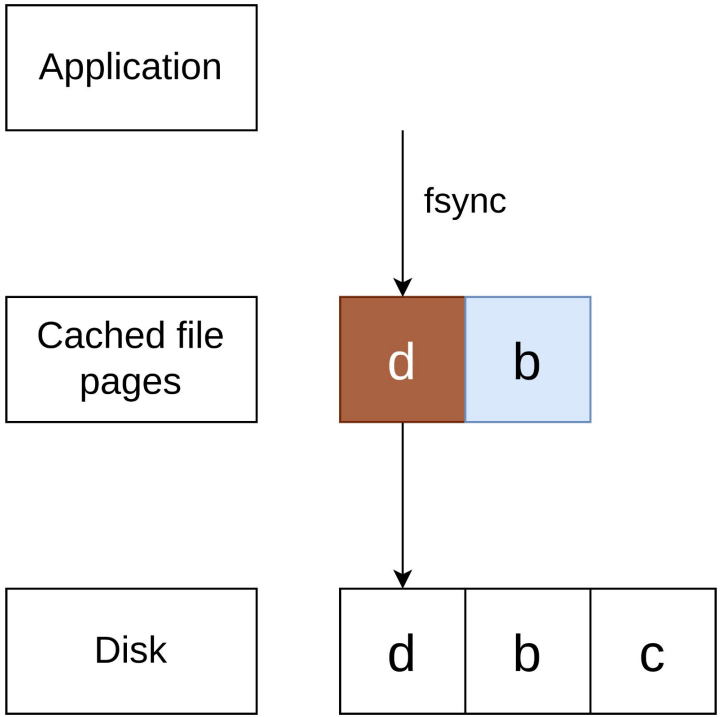
fsyncgate: успешная запись



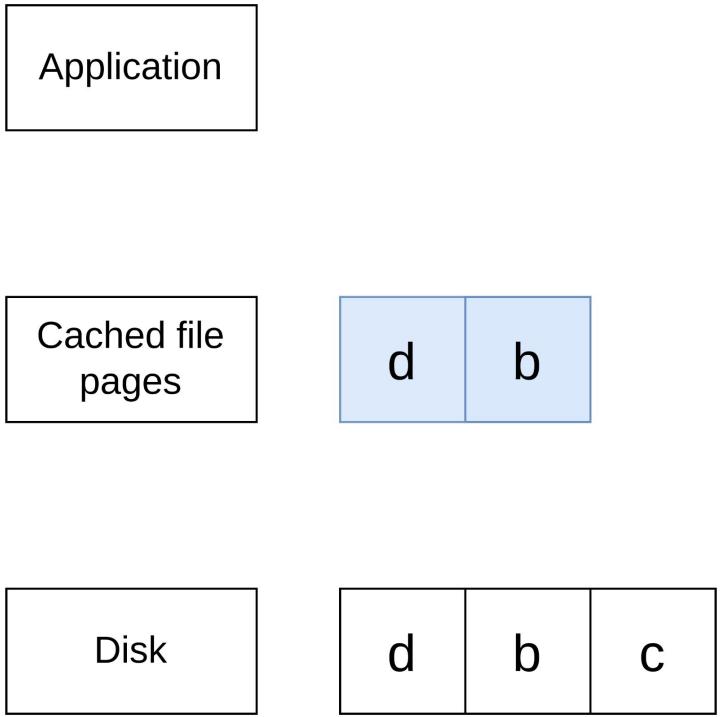
fsyncgate: запись



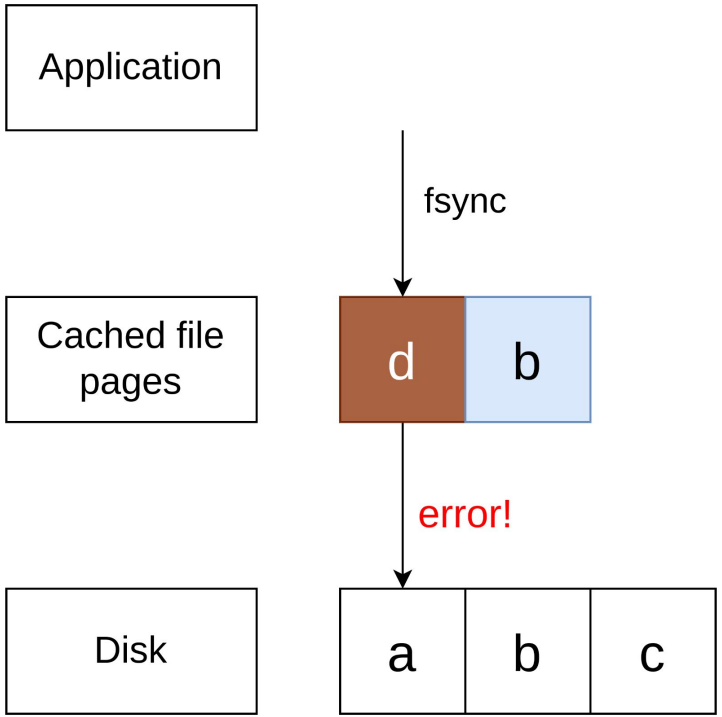
fsyncgate: дергаем fsync



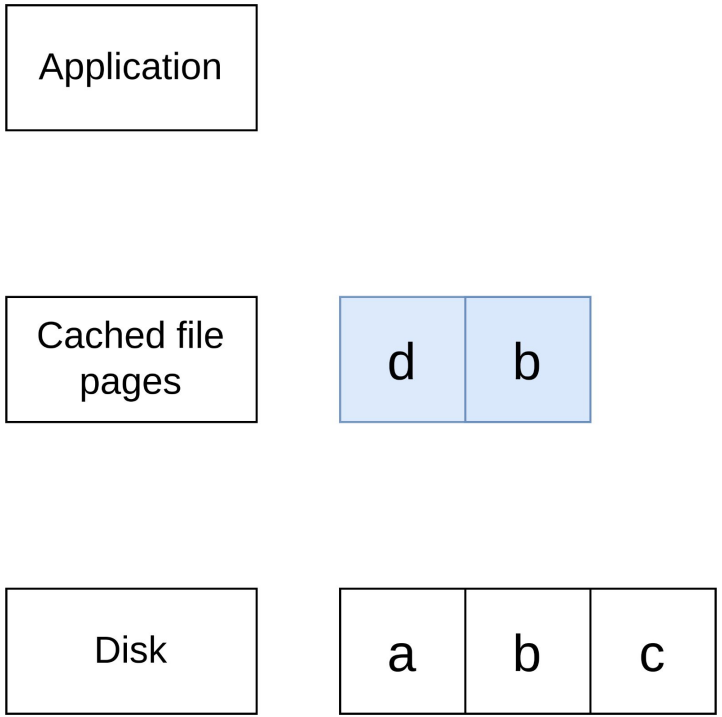
fsyncgate: страница чистая



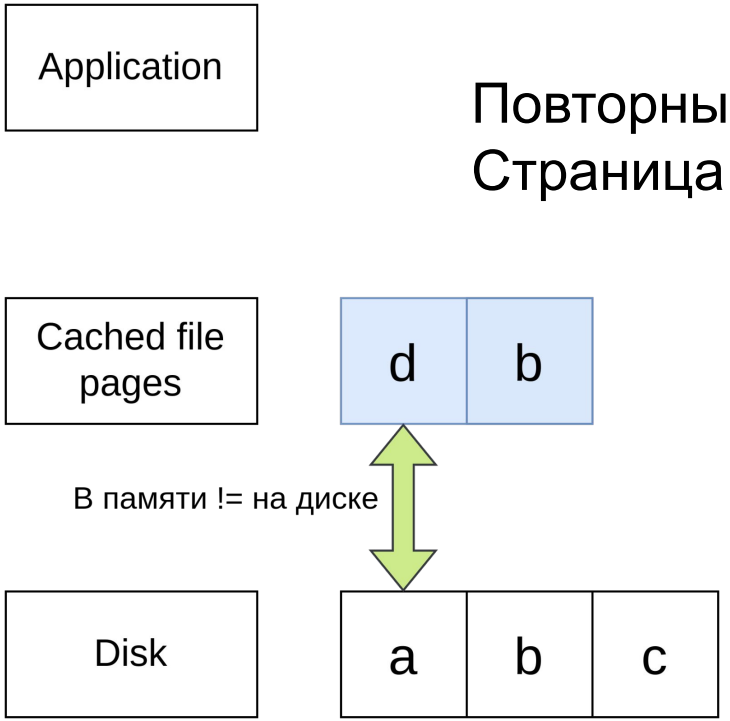
fsyncgate: ошибка fsync



fsyncgate: потерянная запись



fsyncgate: потерянная запись



Повторный вызов — noop!
Страница уже чистая

fsyncgate: итоги

Паника при получении EIO в fsync

fsyncgate: итоги

Паника при получении EIO в fsync

Патчи в ядро на репортинг ошибок в большем количестве случаев

fsyncgate: итоги

Паника при получении EIO в fsync

Патчи в ядро на репортинг ошибок в большем количестве случаев

Горячие споры между разработчиками ядра и разработчиками БД

PostgreSQL Wiki: Fsync Errors

fsyncgate: итоги

If that's actually the case, we need to push back on this kernel brain damage, because as you're describing it fsync would be completely useless.

Tom Lane (Postgres Core Team member)

PostgreSQL Wiki: Fsync Errors

Can Applications Recover from fsync Failures?

- Как fs реагирует на ошибки записи данных во время fsync?
- Какие ошибки приводят к недоступности всей fs? (shutdown/remount-ro)

Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

- Как fs реагирует на ошибки записи данных во время fsync?
- Какие ошибки приводят к недоступности всей fs? (shutdown/remount-ro)

Файловые системы: **ext4, xfs, btrfs**

Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

- Как fs реагирует на ошибки записи данных во время fsync?
- Какие ошибки приводят к недоступности всей fs? (shutdown/remount-ro)

Файловые системы: **ext4, xfs, btrfs**

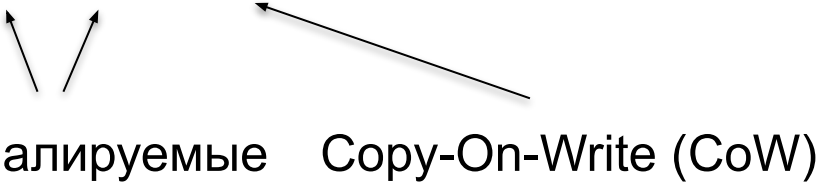
журналируемые

Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

- Как fs реагирует на ошибки записи данных во время fsync?
- Какие ошибки приводят к недоступности всей fs? (shutdown/remount-ro)

Файловые системы: **ext4, xfs, btrfs**



[Can Applications Recover from fsync Failures?]

Can Applications Recover from fsync Failures?

- Как приложения реагируют на ошибки?

Приложения:

Redis, LMDB, LevelDB, SQLite, PostgreSQL, Git

Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

		fsync Failure Basics					Error Reporting		After Effects			Recovery
		Which block failure causes fsync failure?	Is metadata persisted on data block failure?	Which block failures are retried?	Is the page dirty or clean after failure?	Does the in-memory content match disk?	Which fsync reports the failure?	Is the failure logged to syslog?	Which block failure causes unavailability?	What type of unavailability?	Holes or block over-write failures? If yes where do they occur?	Can fsck help detect holes or block over-write failures?
		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
ext4	ordered	data,jrnl	yes [A]		clean [B]	no [B]	immediate	yes	jrnl	remount-ro	NOB, anywhere [A]	no
	data	data,jrnl	yes [A]		clean [B]	no [B]	next [C]	yes	jrnl	remount-ro	NOB, anywhere [A]	no
XFS		data,jrnl	yes [A]	meta	clean [B]	no [B]	immediate	yes	jrnl,meta	shutdown	NOB, within [A]	no
Btrfs		data,jrnl	no		clean	yes	immediate	yes	jrnl,meta	remount-ro	HOLE, within [D]	yes

[Can Applications Recover from fsync Failures?]

Can Applications Recover from fsync Failures?

Страница помечена чистой после ошибки fsync?

- ext4 ordered — **да***
- ext4 data — **да***
- xfs — **да***
- btrfs — **да**

Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

Страница в памяти == страница на диске?

- ext4 ordered — **нет**
- ext4 data — **нет**
- xfs — **нет**
- btrfs — **да**

Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

Какой fsync возвращает ошибку?

- ext4 ordered — **первый**
- ext4 data — **второй**

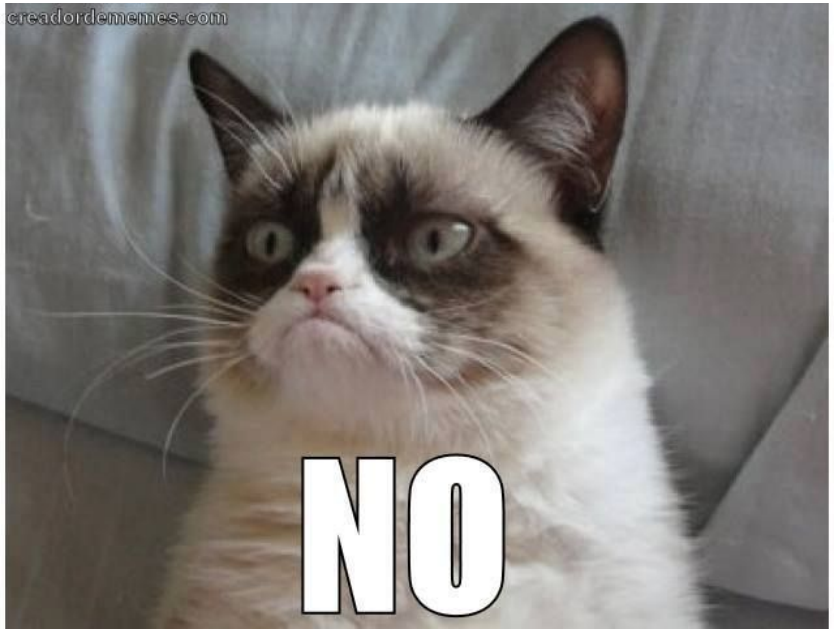
Can Applications Recover from fsync Failures?

Can Applications Recover from fsync Failures?

		fsync Failure Basics					Error Reporting		After Effects			Recovery
		Which block failure causes fsync failure?	Is metadata persisted on data block failure?	Which block failures are retried?	Is the page dirty or clean after failure?	Does the in-memory content match disk?	Which fsync reports the failure?	Is the failure logged to syslog?	Which block failure causes unavailability?	What type of unavailability?	Holes or block over-write failures? If yes where do they occur?	Can fsck help detect holes or block over-write failures?
		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
ext4	ordered data	data,jrnl	yes [A]		clean [B]	no [B]	immediate	yes	jrnl	remount-ro	NOB, anywhere [A]	no
		data,jrnl	yes [A]		clean [B]	no [B]	next [C]	yes	jrnl	remount-ro	NOB, anywhere [A]	no
XFS		data,jrnl	yes [A]	meta	clean [B]	no [B]	immediate	yes	jrnl,meta	shutdown	NOB, within [A]	no
Btrfs		data,jrnl	no		clean	yes	immediate	yes	jrnl,meta	remount-ro	HOLE, within [D]	yes

[Can Applications Recover from fsync Failures?]

Can Applications Recover from fsync Failures?



Почему не могут восстановиться?

- Существующие файловые системы реагируют на ошибки по-разному

Can Applications Recover from fsync Failures?

Почему не могут восстановиться?

- Существующие файловые системы реагируют на ошибки по-разному

  It is reasonable to assert that the key aspects of fsync() are unreasonable to test in a test suite.

Can Applications Recover from fsync Failures? POSIX fsync

Почему не могут восстановиться?

- Существующие файловые системы реагируют на ошибки по-разному

  It is reasonable to assert that the key aspects of fsync() are unreasonable to test in a test suite.

  It would also not be unreasonable to omit testing for fsync(), allowing it to be treated as a quality-of-implementation issue.

Can Applications Recover from fsync Failures? POSIX fsync

Почему не могут восстановиться?

- Существующие файловые системы реагируют на ошибки по-разному

  It is reasonable to assert that the key aspects of fsync() are unreasonable to test in a test suite.

  It would also not be unreasonable to o
  treated as a quality-of-implementation

[Can Applications Recover from fsync Failures?](#) [POSIX fsync](#)

Почему не могут восстановиться?

- Существующие файловые системы реагируют на ошибки по-разному
- Приложения пытаются обработать ошибки fsync каждый по-своему, но этого все равно мало

Can Applications Recover from fsync Failures?

Почему не могут восстановиться?

- Существующие файловые системы реагируют на ошибки по-разному
- Приложения пытаются обработать ошибки fsync каждый по-своему, но этого все равно мало
- Приложения не тестируются с ошибками на уровне отдельных блоков

Can Applications Recover from fsync Failures?

Важность явных контрактов, или Как не потерять данные

append + rename

```
open(tmp);

write(tmp);

close(tmp);

rename(tmp, new);
```

- Распространенный паттерн

POSIX v. reality: A position on O_PONIES

[PATCH] fs: point out any processes using O_PONIES

append + rename

```
open(tmp);

write(tmp);

close(tmp);

rename(tmp, new);
```

- Распространенный паттерн
- ext3 — ФС по умолчанию

POSIX v. reality: A position on O_PONIES

[PATCH] fs: point out any processes using O_PONIES

append + rename

```
open(tmp);

write(tmp);

close(tmp);

rename(tmp, new);
```

POSIX v. reality: A position on O_PONIES

[PATCH] fs: point out any processes using O_PONIES

append + rename

```
open(tmp);

write(tmp);

close(tmp);

rename(tmp, new);
```

- Распространенный паттерн
- ext3 — ФС по умолчанию
- ext4 — отложенная аллокация блоков

[POSIX v. reality: A position on O_PONIES]

[PATCH] fs: point out any processes using O_PONIES

append + rename

```
open(tmp);

write(tmp);

close(tmp);

rename(tmp, new);
```

А контракты — ненастоящие!

POSIX v. reality: A position on O_PONIES

[PATCH] fs: point out any processes using O_PONIES

Как надежно переименовать файл?

durable_rename in file_utils.h

Как надежно переименовать файл?

fsync(old)

durable_rename in file_utils.h

Как надежно переименовать файл?

fsync(old)

fsync(new)

durable_rename in file_utils.h

Как надежно переименовать файл?

fsync(old)

fsync(new)

rename(old, new)

durable_rename in file_utils.h

Как надежно переименовать файл?

fsync(old)

fsync(new)

rename(old, new)

fsync(new)

durable_rename in file_utils.h

Как надежно переименовать файл?

fsync(old)

fsync(new)

rename(old, new)

fsync(new)

fsync(parent_new)

durable_rename in file_utils.h

Как тестировать?

Взять да отключить!

I've connected the system to a sophisticated power-loss-making device called "the power switch" (image attached).

silent data loss with ext4 / all current versions

Failure injection for fun and profit

- Проблемы зачастую возникают в коде обработки ошибок

Failure injection for fun and profit

- Проблемы зачастую возникают в коде обработки ошибок
- Обработка ошибок обычно хуже тестируется

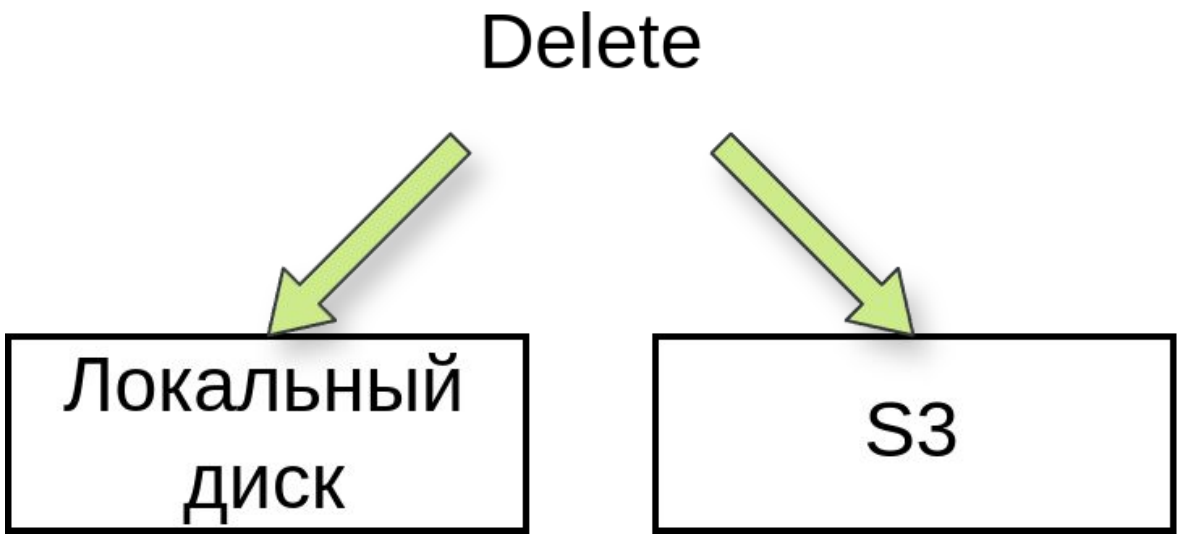
Failure injection for fun and profit

- Проблемы зачастую возникают в коде обработки ошибок
- Обработка ошибок обычно хуже тестируется

```
if failure_enabled("foo"):
    raise Exception("boom!")
```

Failure injection: case study

Задача:



Failure injection: clean delete

- Удалить данные из двух мест (много)
- Оба этапа неатомарные
- После начала удаления запретить использование
- Чистое завершение, несмотря на рестарты и ошибки

Как удаляем: подготовка

- Set deleted_at in remote index part
- Create local mark file

Как удаляем: удаление

- Set deleted_at in remote index part
- Create local mark file
- Delete local files except mark file
- Delete remote files

Как удаляем: чистим следы

- Set deleted_at in remote index part
- Create local mark file
- Delete local files except mark file
- Delete remote files
- Delete remote index part
- Delete remaining local directories, delete mark file

Подводные камни:

- Ошибки на каждом шагу (сеть, диск)

Подводные камни:

- Ошибки на каждом шагу (сеть, диск)
- Рестарты сервера, повторные запросы после ошибок

Подводные камни:

- Ошибки на каждом шагу (сеть, диск)
- Рестарты сервера, повторные запросы после ошибок
- Поддержать возобновление с любого шага

Failure injection setup

- На каждое действие, которое возвращает ошибку, добавить выбрасывание ошибки по команде
- Написать тест, который по отдельности включит каждую ошибку, и проверить выполнение требований

Тестим

```
@pytest.mark.parametrize("failpoint", [...])
@pytest.mark.parametrize("restart_or_retry", ["restart", "retry"])
def test_delete(failpoint, restart_or_retry):
    setup(...)
    enable_failpoint(failpoint)

    error = delete()
    assert error == ...

    if restart_or_retry == "restart":
        restart()
    else:
        delete()

    assert delete_success()
```

Тестим

```
@pytest.mark.parametrize("failpoint", [...])
@pytest.mark.parametrize("restart_or_retry", ["restart", "retry"])
def test_delete(failpoint, restart_or_retry):
    setup(...)
    enable_failpoint(failpoint)

    error = delete()
    assert error == ...

    if restart_or_retry == "restart":
        restart()
    else:
        delete()

    assert delete_success()
```

Тестим

```
@pytest.mark.parametrize("failpoint", [...])
@pytest.mark.parametrize("restart_or_retry", ["restart", "retry"])
def test_delete(failpoint, restart_or_retry):
    setup(...)
    enable_failpoint(failpoint)

    error = delete()
    assert error == ...

    if restart_or_retry == "restart":
        restart()
    else:
        delete()

    assert delete_success()
```

Тестим

```
@pytest.mark.parametrize("failpoint", [...])
@pytest.mark.parametrize("restart_or_retry", ["restart", "retry"])
def test_delete(failpoint, restart_or_retry):
    setup(...)
    enable_failpoint(failpoint)

    error = delete()
    assert error == ...

    if restart_or_retry == "restart":
        restart()
    else:
        delete()

    assert delete_success()
```

Тестим

```
@pytest.mark.parametrize("failpoint", [...])
@pytest.mark.parametrize("restart_or_retry", ["restart", "retry"])
def test_delete(failpoint, restart_or_retry):
    setup(...)
    enable_failpoint(failpoint)

    error = delete()
    assert error == ...

    if restart_or_retry == "restart":
        restart()
    else:
        delete()

    assert delete_success()
```

Тестим

```
@pytest.mark.parametrize("failpoint", [...])
@pytest.mark.parametrize("restart_or_retry", ["restart", "retry"])
def test_delete(failpoint, restart_or_retry):
    setup(...)
    enable_failpoint(failpoint)

    error = delete()
    assert error == ...

    if restart_or_retry == "restart":
        restart()
    else:
        delete()

    assert delete_success()
```

Failure injection: результат

- Таким нехитрым способом нашлось 5 ошибок в нашем коде

Failure injection: результат

- Таким нехитрым способом нашлось 5 ошибок в нашем коде
- Большинство связаны рестартами

Failure injection: результат

- Таким нехитрым способом нашлось 5 ошибок в нашем коде
- Большинство связаны рестартами
- Нужно модифицировать код

Failure injection: результат

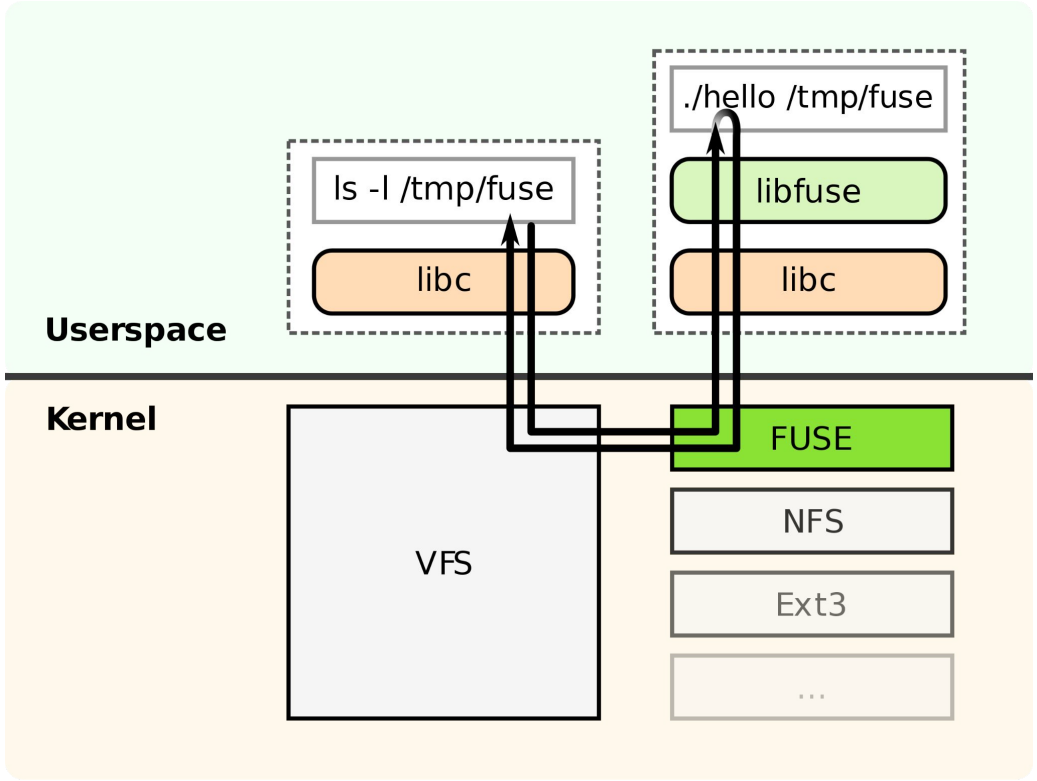
- Таким нехитрым способом нашлось 5 ошибок в нашем коде
- Большинство связаны рестартами
- Нужно модифицировать код

Но как быть с ошибками на уровне ФС?

Внедрение ошибок на уровне ФС

- Нужно заставить ФС нам врать
- ФС — куча кода в ядре

FUSE приходит на помощь



Интересные факты о FUSE

- Применяется в системах контроля версий, ориентированных на монорепы (Яндексовая Аркадия AFAIK, Facebook Sapling)
- Реализации sshfs, s3fs
- Исследование производительности FUSE: To FUSE or Not to FUSE: Performance of User-Space File Systems
- RFUSE: Modernizing Userspace Filesystem Framework through Scalable Kernel-Userspace Communication

FUSE для тестирования

Возврат ошибок в системных вызовах (read, write, fsync)

Терять данные, которые не были синхронизированы

Частично записывать данные

Ссылки:

unreliablefs

lazyfs (Request for a Lazy Filesystem)

FUSE для тестирования

- Полностью локально

FUSE для тестирования

- Полностью локально
- Не нужно изменять код приложения

FUSE для тестирования

- Полностью локально
- Не нужно изменять код приложения
- Внедрение ошибок на конкретные операции вместо случайных падений

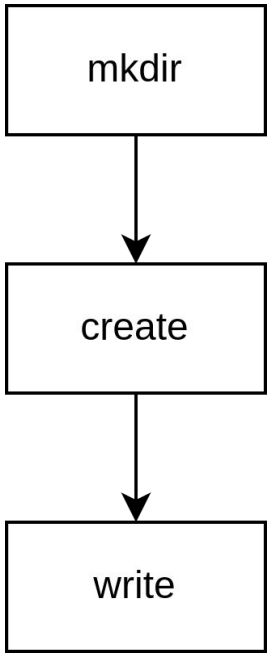
FUSE для тестирования

- Полностью локально
- Не нужно изменять код приложения
- Внедрение ошибок на конкретные операции вместо случайных падений
- Воспроизводимость лучше, но все равно не гарантируется

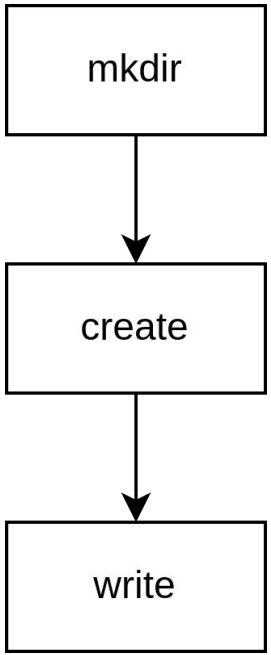
FUSE для тестирования

- Полностью локально
- Не нужно изменять код приложения
- Внедрение ошибок на конкретные операции вместо случайных падений
- Воспроизводимость лучше, но все равно не гарантируется
- ~Платформозависимость

Проблема состояний

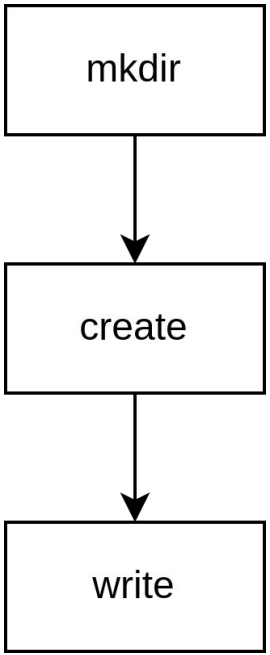


Проблема состояний



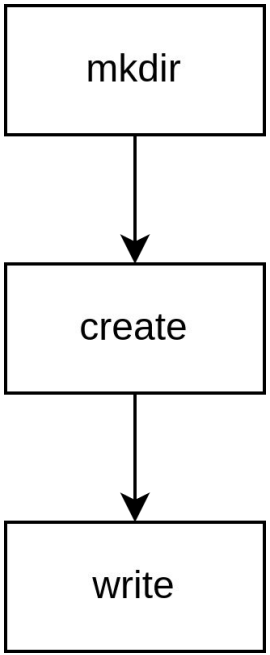
- Директория не доехала до диска

Проблема состояний



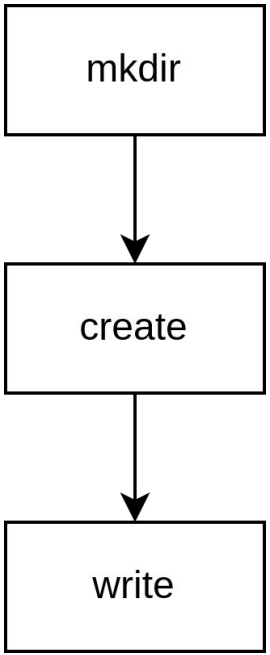
- Директория не доехала до диска
- Создание файла не доехало до диска

Проблема состояний



- Директория не доехала до диска
- Создание файла не доехало до диска
- Записанные данные не доехали

Проблема состояний



- Директория не доехала до диска
- Создание файла не доехало до диска
- Записанные данные не доехали

Тестировать перебором состояний

ALICE

All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

ALICE

Workload:

```
f = open(“f")
f.write("stuff")
print("ok")
```

[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

ALICE

Workload:

```
f = open(“f")
f.write("stuff")
print("ok")
```

Checker:

```
if output.contains("ok"):
    f = open("f")
    assert f.contains("stuff")
```

[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

ALICE

Workload:

```
f = open(“f")
f.write("stuff")
print("ok")
```

} strace

Checker:

```
if output.contains("ok"):
    f = open("f")
    assert f.contains("stuff")
```

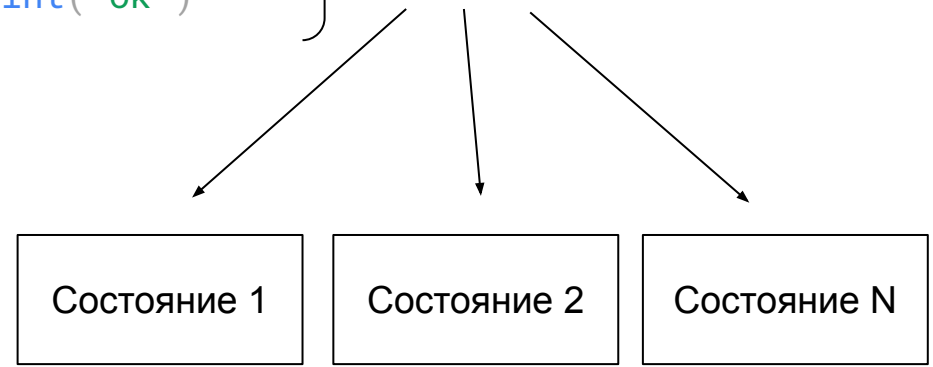
All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

ALICE

Workload:

```
f = open(“f")
f.write("stuff")
print("ok")
```

strace



Checker:

```
if output.contains("ok"):
    f = open("f")
    assert f.contains("stuff")
```

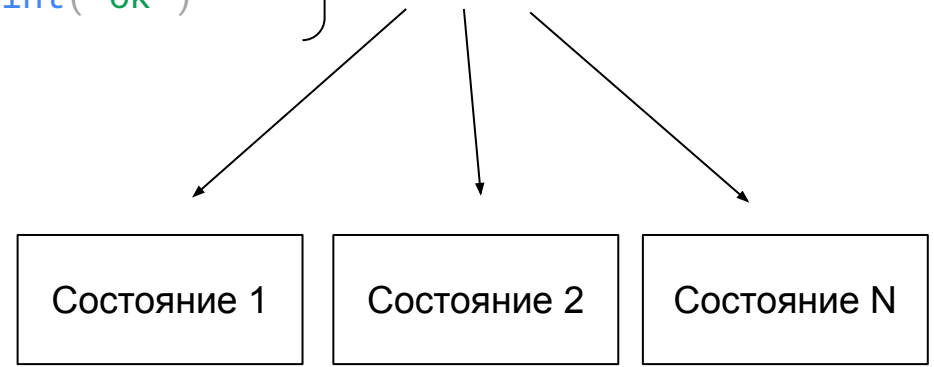
[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

ALICE

Workload:

```
f = open(“f")
f.write("stuff")
print("ok")
```

strace



Checker:

```
if output.contains("ok"):
    f = open("f")
    assert f.contains("stuff")
```

[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

ALICE

- Лучше покрытие

All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

ALICE

- Лучше покрытие
- Код менять не нужно

All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы

All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы

- Не все возможности есть в опубликованной версии

[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы

- Не все возможности есть в опубликованной версии
- Не поддерживается

[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы

- Не все возможности есть в опубликованной версии
- Не поддерживается
- Использование strace несовместимо с io_uring

All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы

- Не все возможности есть в опубликованной версии
- Не поддерживается
- Использование strace несовместимо с io_uring

Итог: можно ограниченно использовать

[All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications]

Чего еще нам не хватает?

- POSIX формальнее не становится

Чего еще нам не хватает?

- POSIX формальнее не становится
- Тесты показывают наличие проблем, а не их отсутствие

Чего еще нам не хватает?

- POSIX формальнее не становится
- Тесты показывают наличие проблем, а не их отсутствие

Формальные методы!

Формальные методы

- Математическое доказательство корректности

Формальные методы

- Математическое доказательство корректности
  - Долго
  - Дорого

Путь в светлое будущее и другие исследования по теме

- Применение опыта верификации моделей памяти

Specifying and Checking File System Crash-Consistency Models

Путь в светлое будущее и другие исследования по теме

- Применение опыта верификации моделей памяти
  - Автоматическая вставка нужных fsync, чтоб стало как надо (тм)

Specifying and Checking File System Crash-Consistency Models

Путь в светлое будущее и другие исследования по теме

- Применение опыта верификации моделей памяти
  - Автоматическая вставка нужных fsync, чтоб стало как надо (тм)
- Разработка верифицированных файловых систем

  FSCQ is the first file system with a machine-checkable proof (using the Coq proof assistant) that its implementation meets its specification and whose specification includes crashes.

Filesystem error handling

Using Crash Hoare Logic for Certifying the FSCQ File System

Верификация ФС — не панацея

- Не поможет неверифицированным приложениям

Верификация ФС — не панацея

- Не поможет неверифицированным приложениям
- Переход на API, которые невозможно неправильно использовать

Итог

- Все сломано

Итог

- Все сломано
- Пишите тесты

Итог

- Все сломано
- Пишите тесты
- Использовать проверенное

Итог

- Все сломано
- Пишите тесты
- Использовать проверенное
- Изобретать свое — со знанием дела

Итог

- Все сломано
- Пишите тесты
- Использовать проверенное
- Изобретать свое — со знанием дела
- Делать бэкапы

Голосуйте за мой доклад

Дмитрий Родионов

@LizardWizzard (telegram/github)

<todo ссылка на слайды>

HL 2024 HighLoad++

Библиография: 1

- Clarifying Direct IO's Semantics
- Ensuring data reaches disk
- Coerced cache eviction and discreet mode journaling: Dealing with misbehaving disks
- man 2 fsync
- man 2 close
- PostgreSQL's handling of fsync() errors is unsafe and risks data loss at least on XFS
- PostgreSQL Wiki: Fsync Errors

Библиография: 2

- Can Applications Recover from fsync Failures?
- POSIX fsync
- POSIX v. reality: A position on O_PONIES
- [PATCH] fs: point out any processes using O_PONIES
- durable_rename in file_utils.h
- All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications (slides)
- man fsync

Библиография: 3

- All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications
- Protocol-Aware Recovery for Consensus-Based Storage
- silent data loss with ext4 / all current versions
- Durability and Redo Logging
- To FUSE or Not to FUSE: Performance of User-Space File Systems
- RFUSE: Modernizing Userspace Filesystem Framework through Scalable Kernel-Userspace Communication

Библиография: 4

- unreliablefs
- lazyfs
- Request for a Lazy Filesystem
- All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications
- Specifying and Checking File System Crash-Consistency Models
- Filesystem error handling
- Using Crash Hoare Logic for Certifying the FSCQ File System
- Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems