

Файлы это сложно

Кто виноват и что делать?

О себе



- Занимаюсь базами данных и распределенными системами
- Ведущий разработчик в [Picodata](#)
- Аспирант в ИСП РАН

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

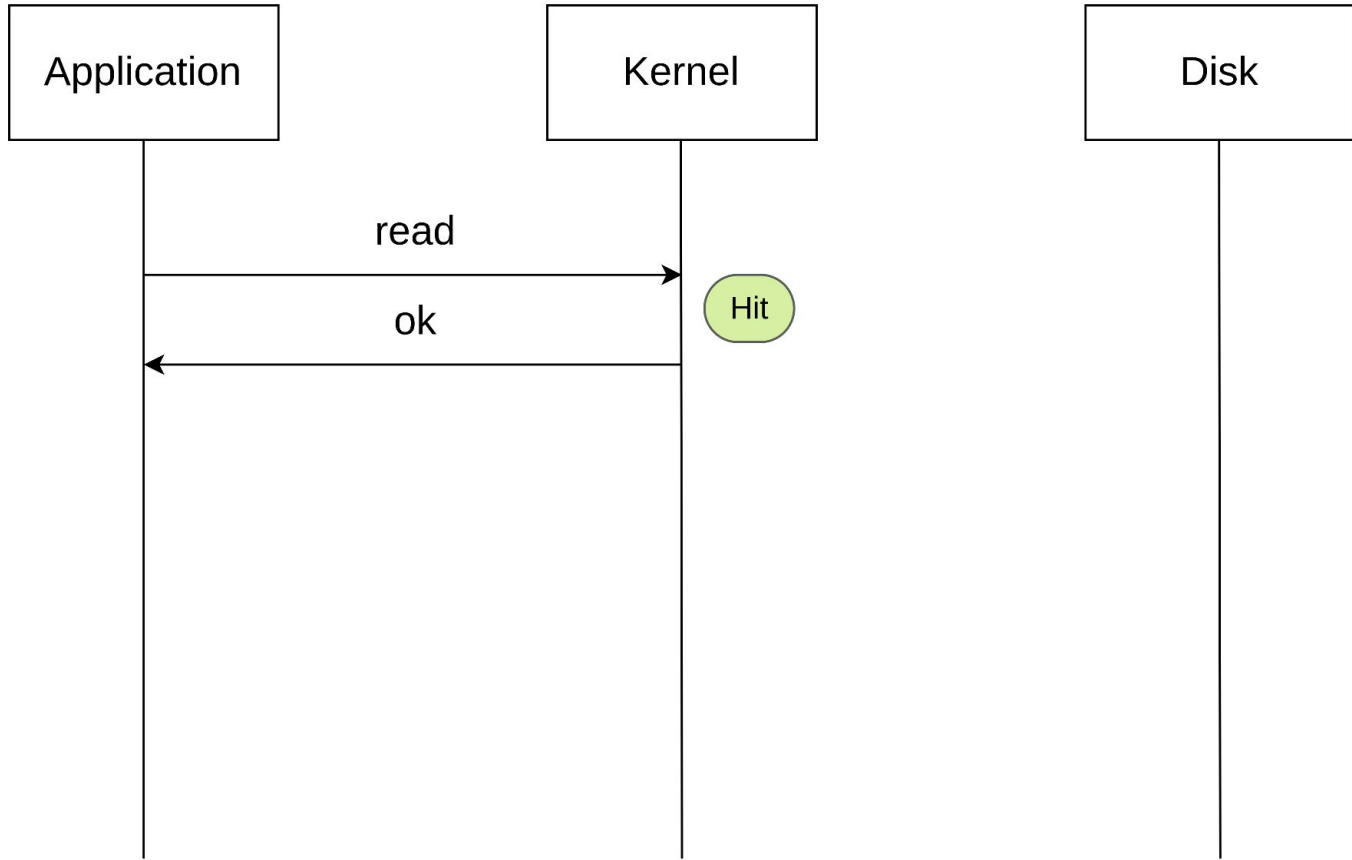
Что сложного? Берем да записываем

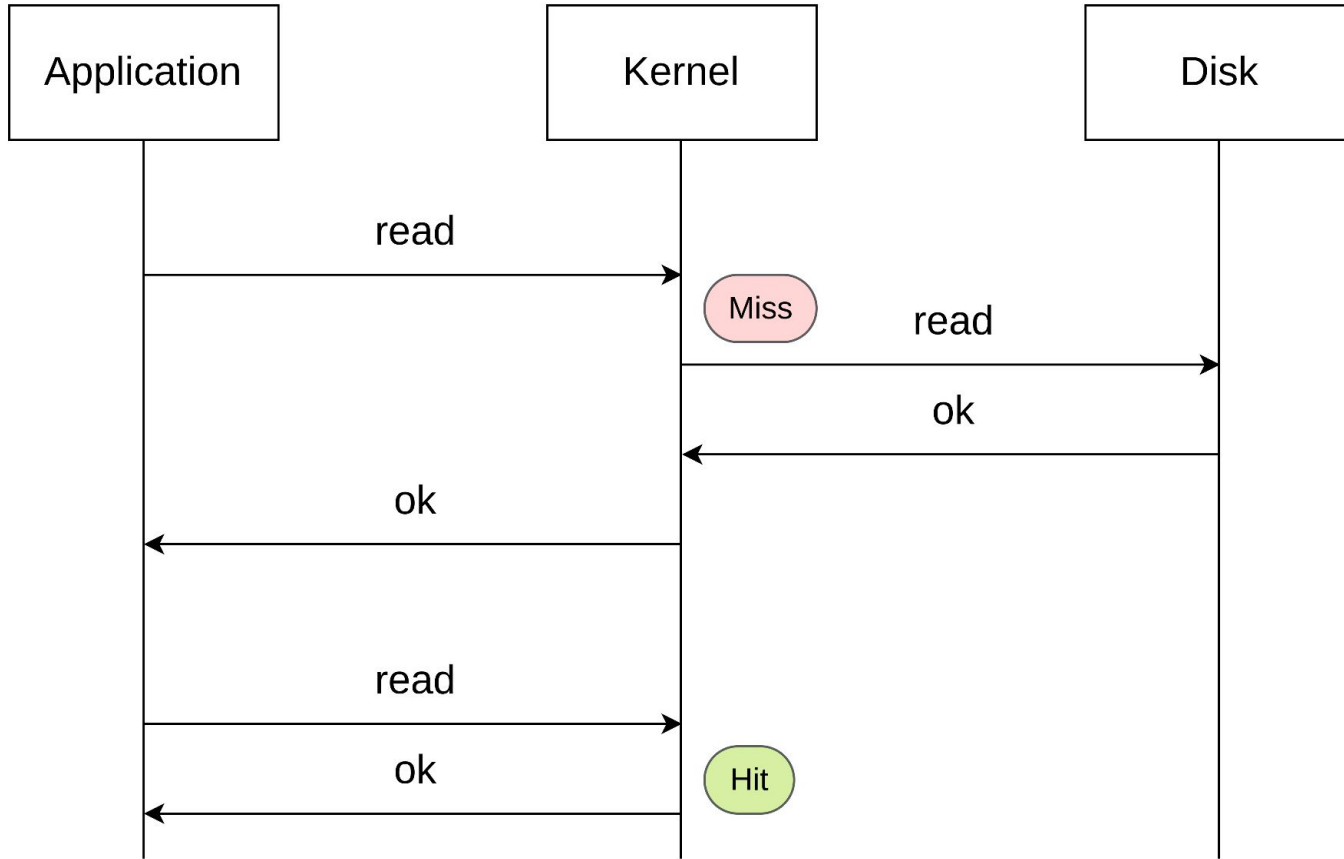
```
let f = File::create("hello");
```

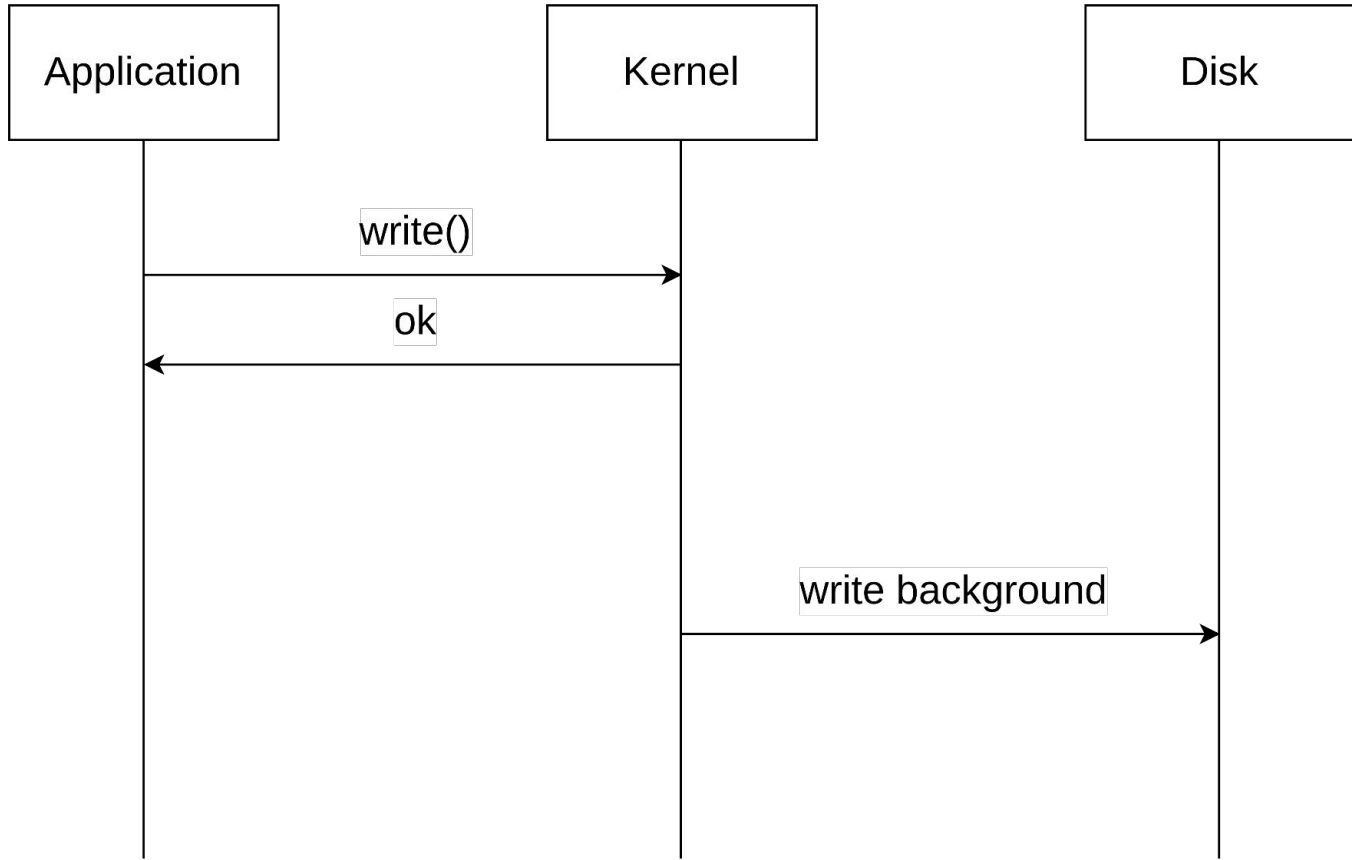
```
let buf = b"hey";
```

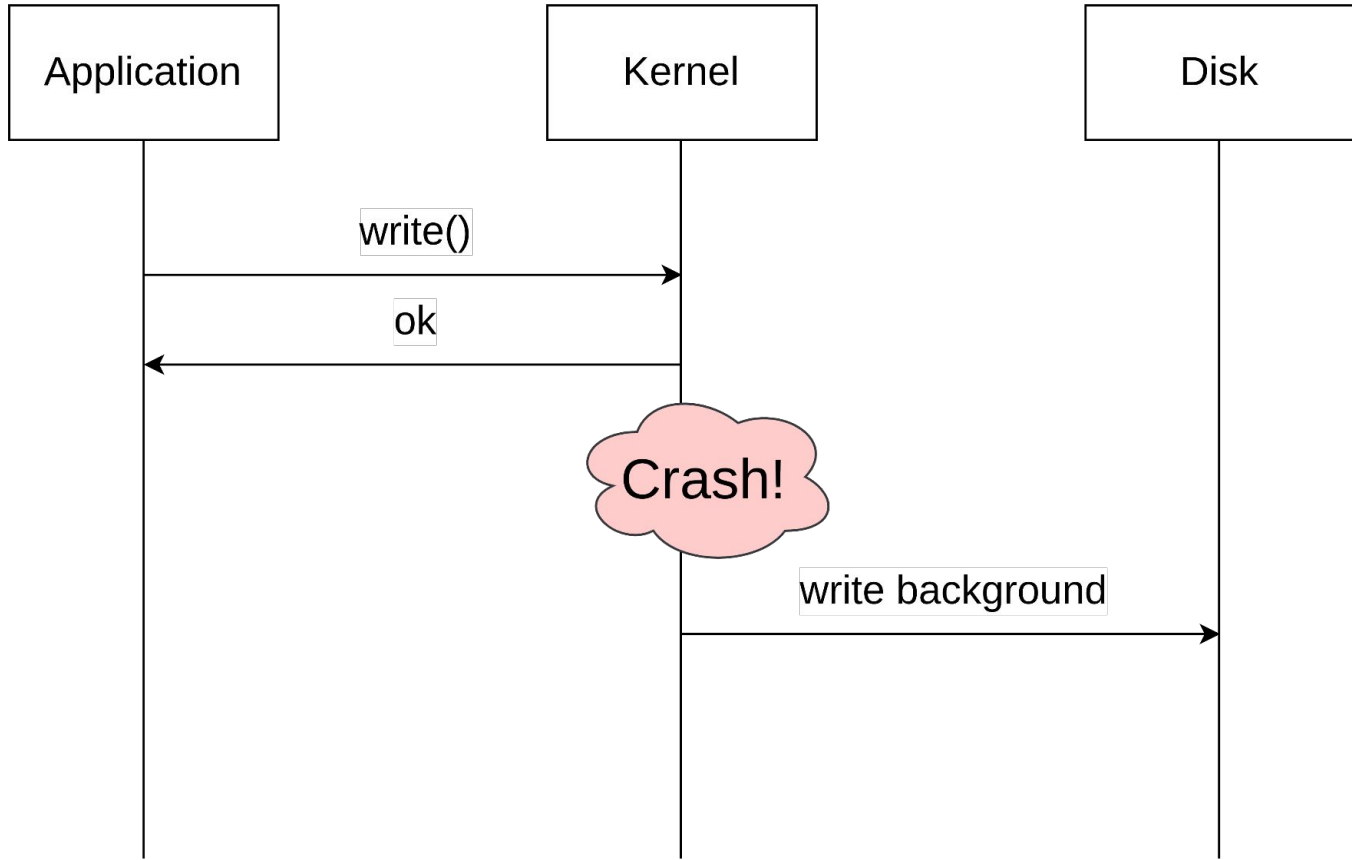
```
f.write_at(buf, 0);
```

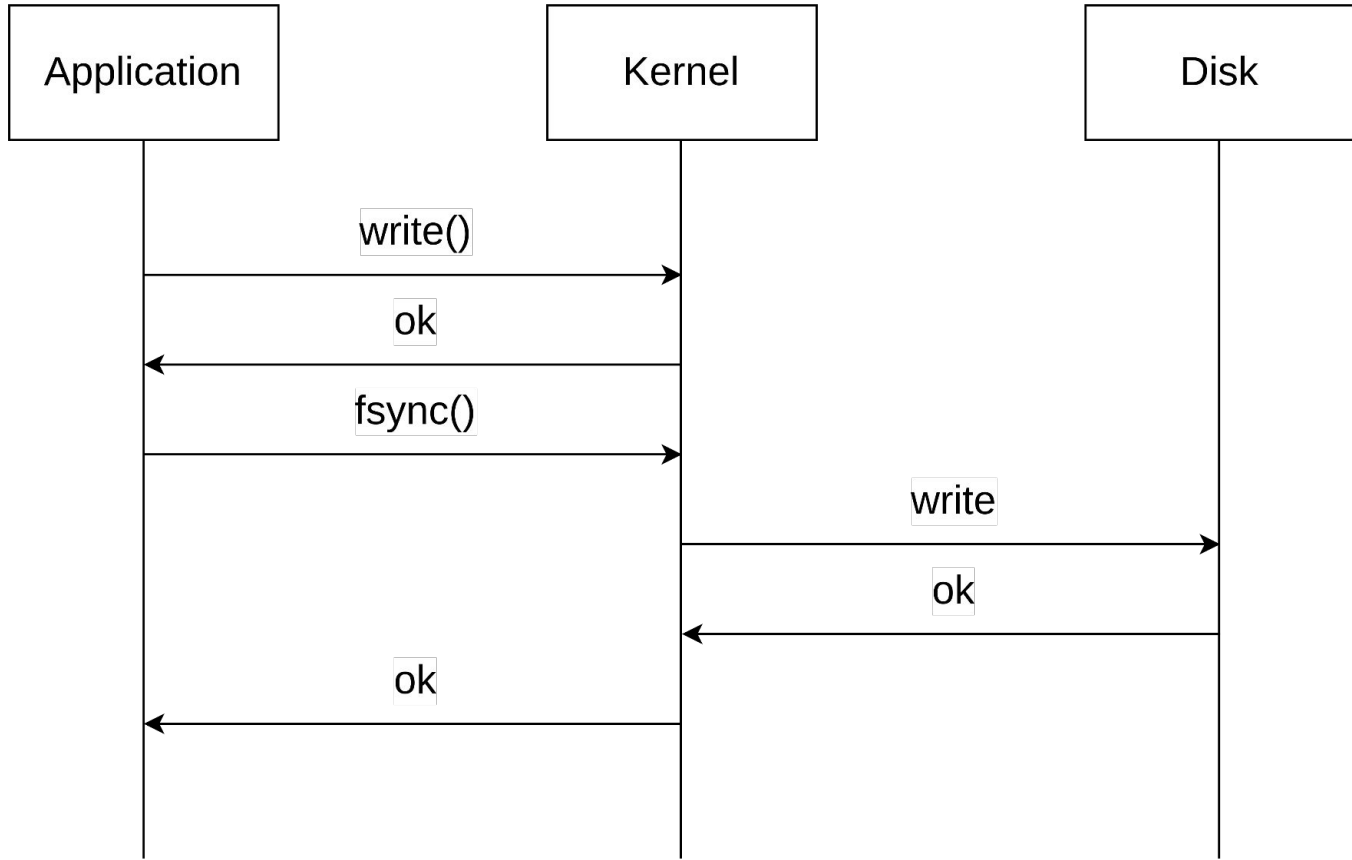
Ввод-вывод для блочных устройств

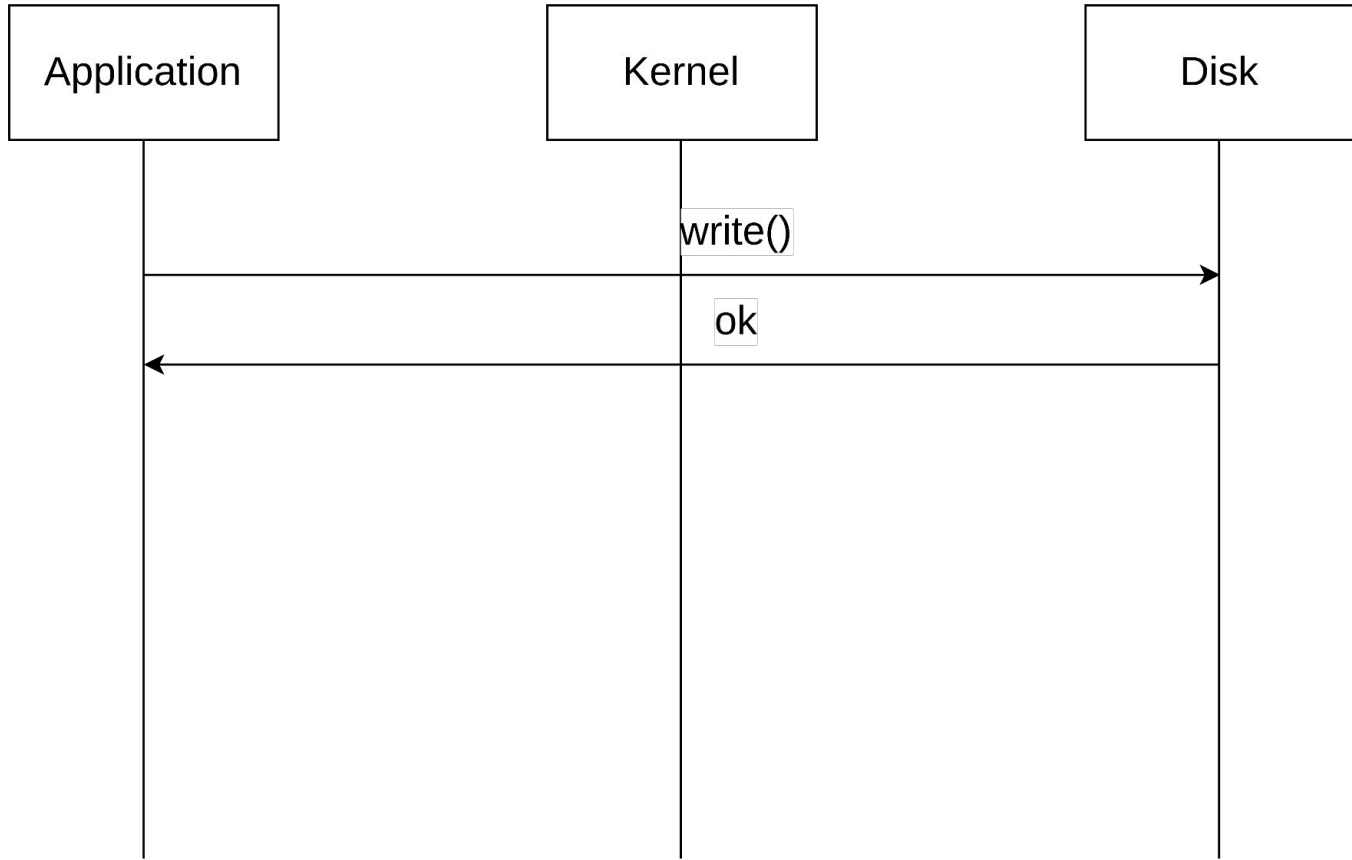












Немного истории O_DIRECT

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

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

fsync и O_DIRECT

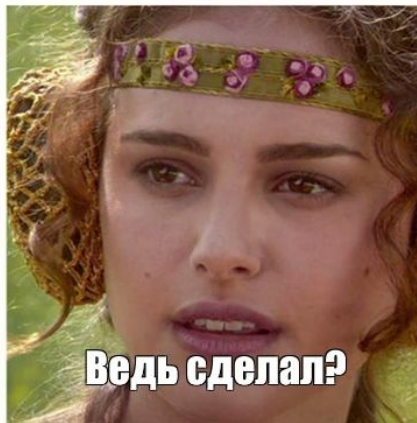
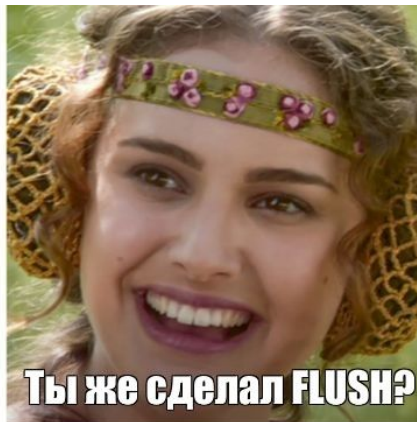
fsync и O_DIRECT

- Диск имеет собственный write cache

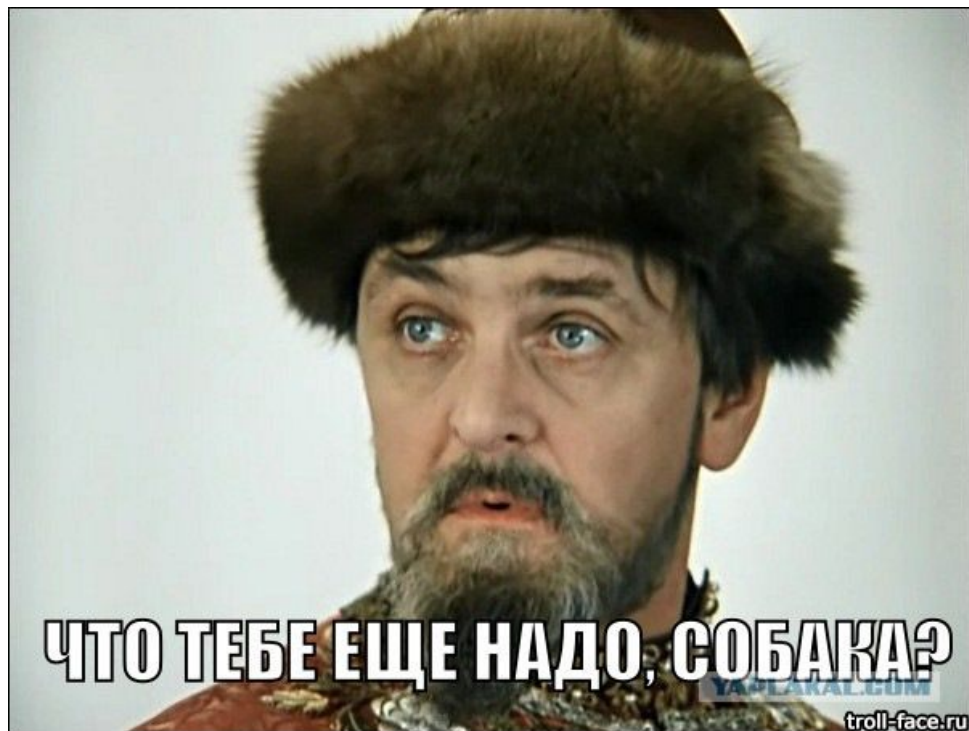
fsync и O_DIRECT

- Диск имеет собственный write cache
- Запись метаданных при выделении новых блоков

Ну теперь-то все хорошо?



Ну сейчас точно все хорошо!



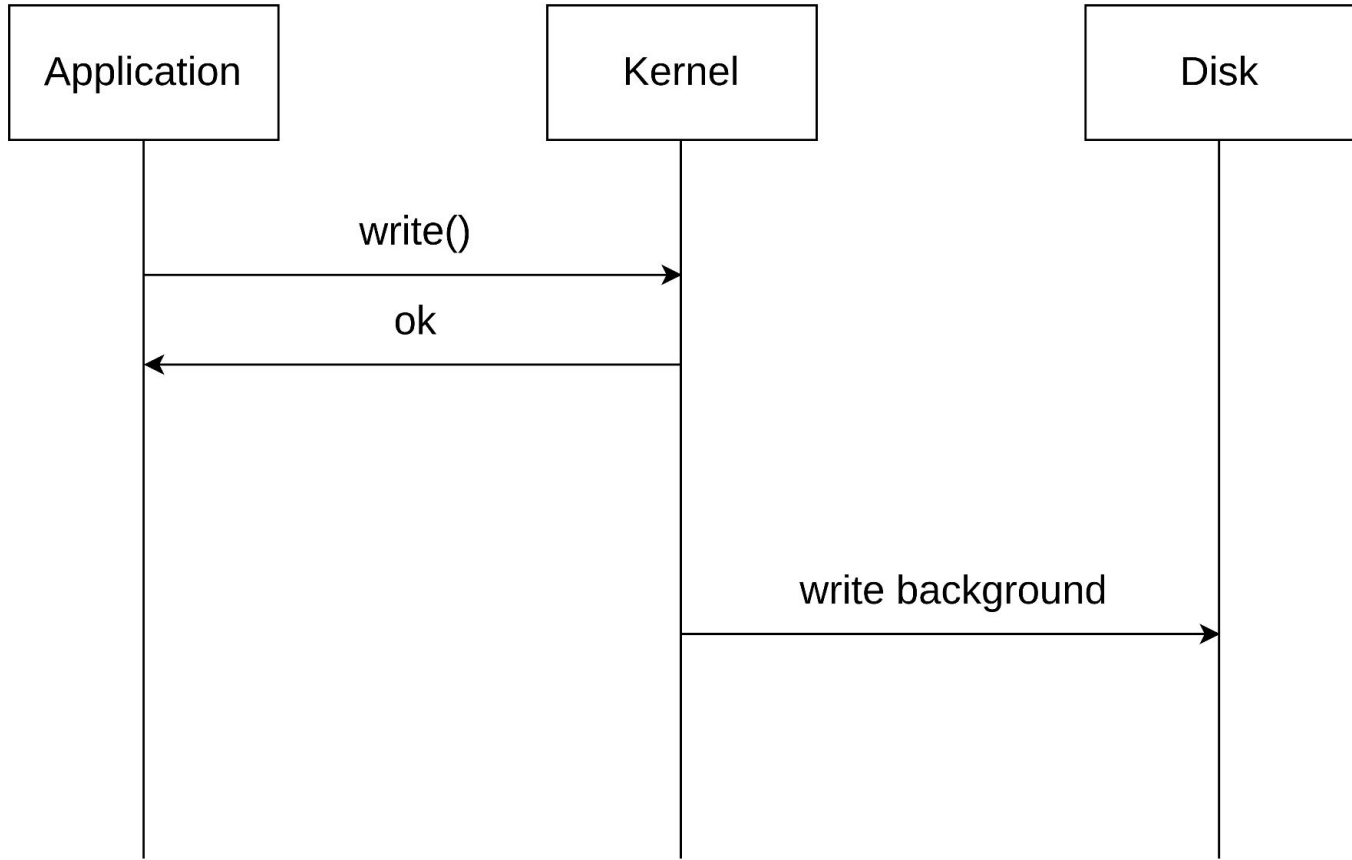
Нельзя просто так взять

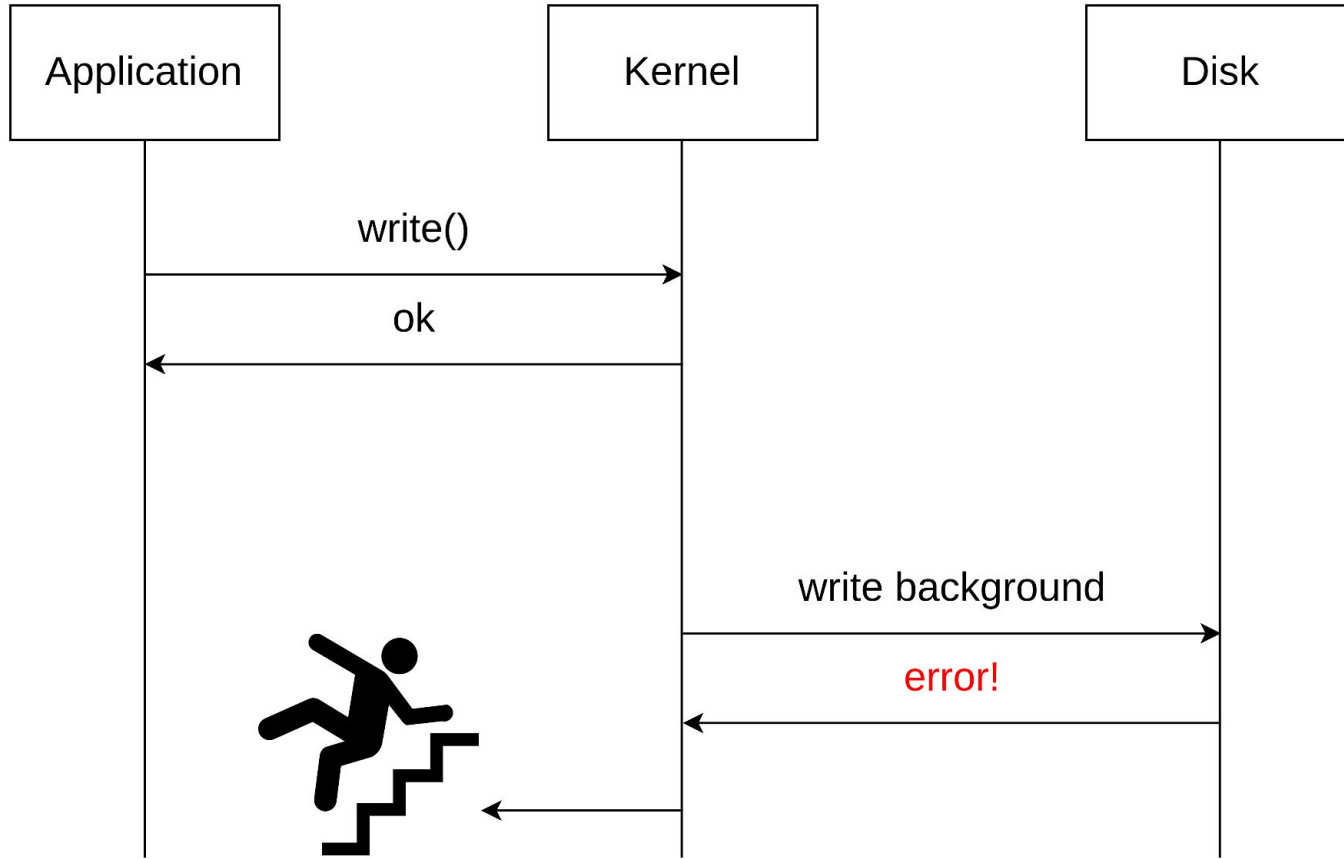
и позвать fsync на Mac OS

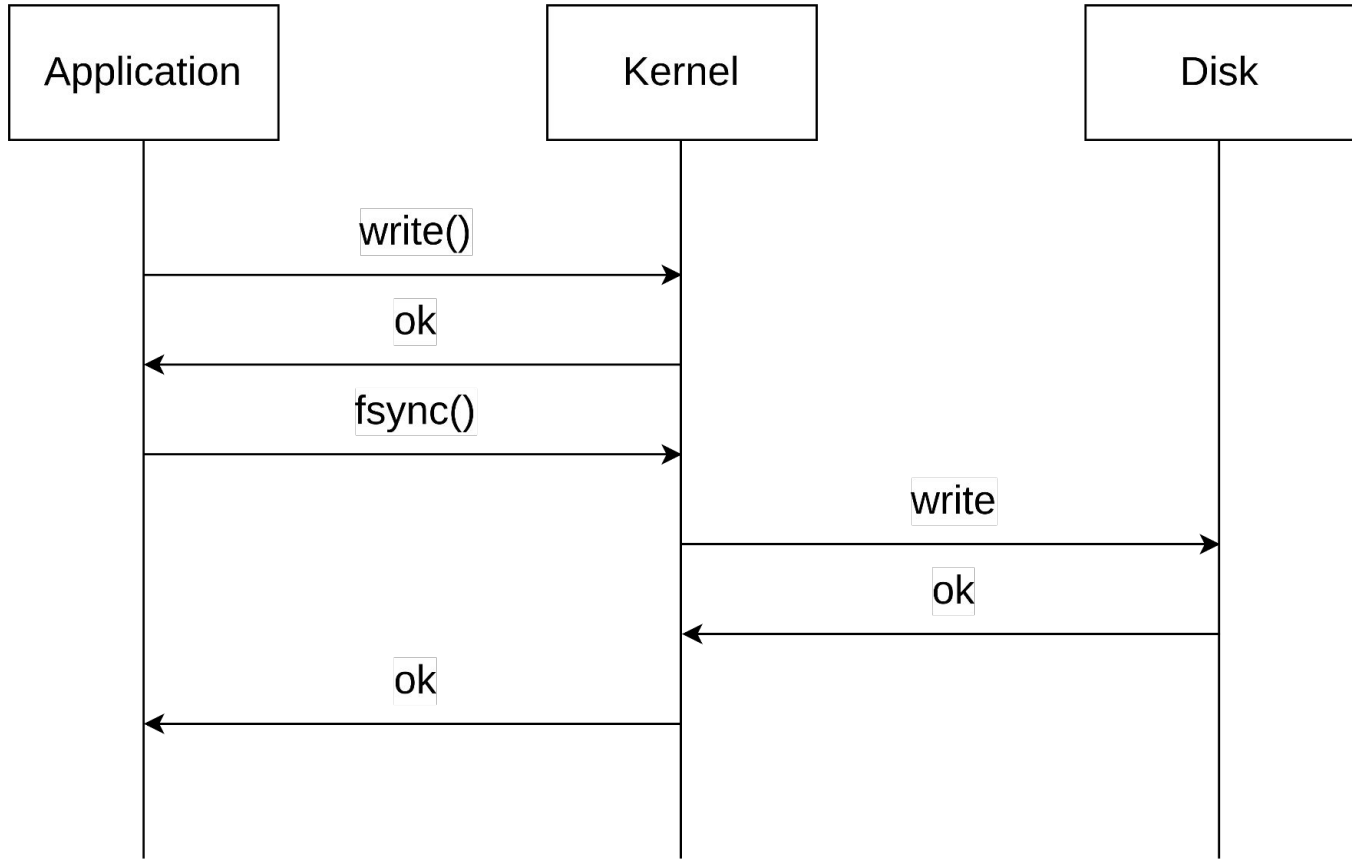
F_FULLFSYNC

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)`







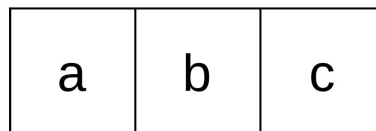
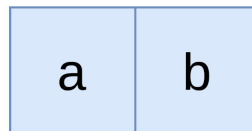
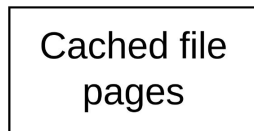
fsyncgate

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

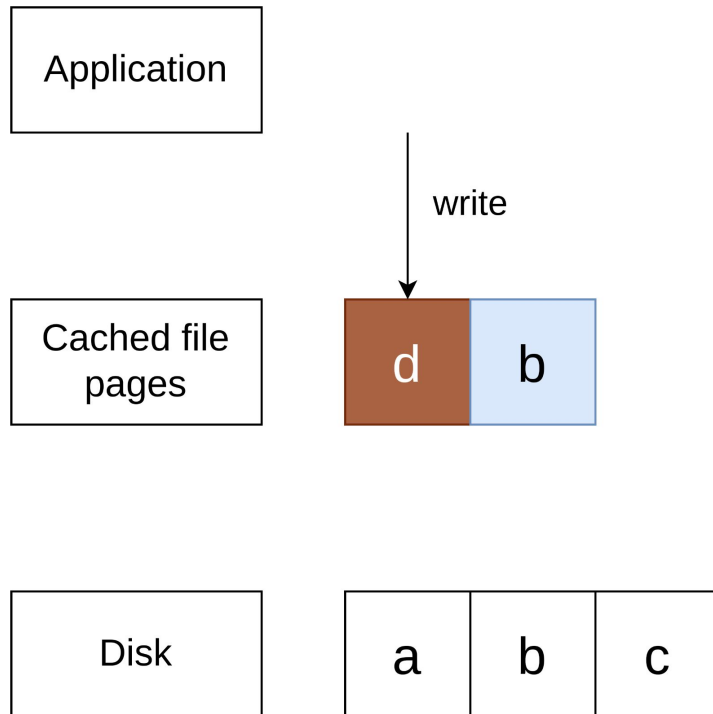
Lists: [pgsql-hackers](#)

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

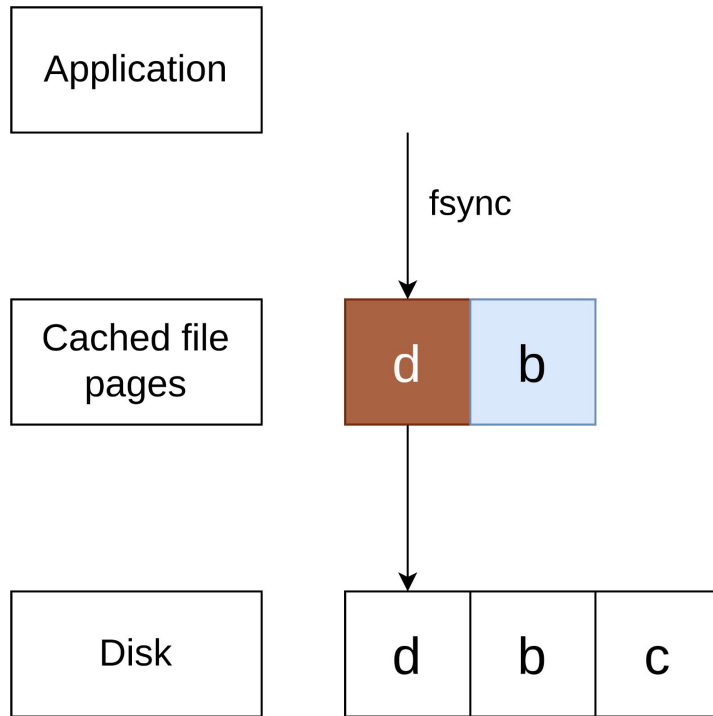
fsyncgate



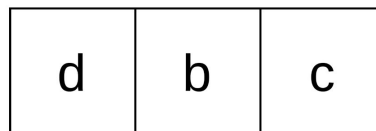
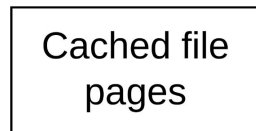
fsyncgate



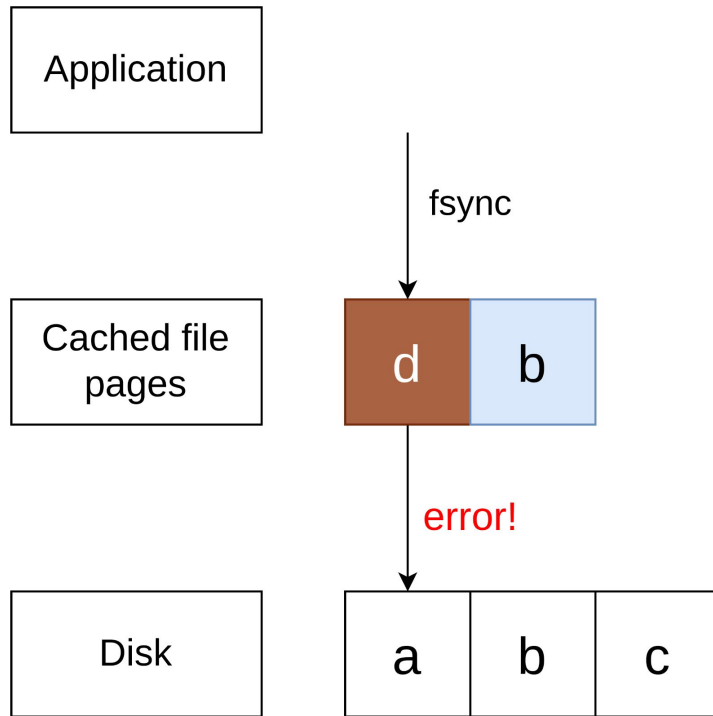
fsyncgate



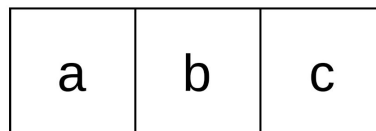
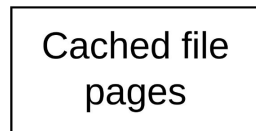
fsyncgate



fsyncgate



fsyncgate



fsyncgate

fsyncgate



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:

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

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)
- Как приложения реагируют на ошибки?

ext4, xfs, btrfs

Redis, LMDB, LevelDB, SQLite, PostgreSQL

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?

Can Applications Recover from fsync Failures?

- Redis: код возврата fsync [не проверялся](#) (актуальное копать [тут](#))

Can Applications Recover from fsync Failures?

- Redis: код возврата fsync [не проверялся](#) (актуальное копать [тут](#))
- LevelDB: данные после ошибки записи оставались доступными

Так могут или нет?

- Существующие файловые системы реагируют на ошибки по-разному

Так могут или нет?

- 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.

Так могут или нет?

- 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.

Так могут или нет?

- 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.



Так могут или нет?

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

Так могут или нет?

- 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

Так могут или нет?

- Существующ
- Приложения
- Приложения



блоки по-разному
ий по своему но
дельных блоков

append + rename

```
open(tmp);
```

```
write(tmp);
```

```
close(tmp);
```

```
rename(tmp, old);
```

append + rename

```
open(tmp);
```

```
write(tmp);
```

```
close(tmp);
```

```
rename(tmp, old);
```

Как надёжно переименовать файл?

`fsync(old)`

`fsync(new)`

`rename(old, new)`

`fsync(new)`

`fsync(parent_new)`

Атомарная перезапись

/x/f1	а		ф	о	о
-------	---	--	---	---	---

Атомарная перезапись

/x/f1	a		f	o	o
-------	---	--	---	---	---

/x/f1	a		b	a	r
-------	---	--	---	---	---

Атомарная перезапись

/x/f1	a		f	o	o
-------	---	--	---	---	---

`pwrite(/x/f1, 2, "bar")`

/x/f1	a		b	a	r
-------	---	--	---	---	---

Атомарная перезапись

/x/f1	a		f	o	o
-------	---	--	---	---	---

/x/f1	a		b	o	o
-------	---	--	---	---	---

`pwrite(/x/f1, 2, "bar")`

/x/f1	a		b	a	r
-------	---	--	---	---	---

Атомарная перезапись

/x/f1	a		f	o	o
-------	---	--	---	---	---

/x/f1	a		b	o	o
-------	---	--	---	---	---

`pwrite(/x/f1, 2, "bar")`

/x/f1	a		b	a	r
-------	---	--	---	---	---

/x/f1	a		f	a	r
-------	---	--	---	---	---

Атомарная перезапись: Undo logging

- Скопировать старые данные в отдельный “лог” файл
- Модифицировать файл
- Удалить лог
- При потере питания/ошибке на этапе модификации восстанавливаем данные из лога

Атомарная перезапись: v1

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, foo");
```



<offset, size, data>

```
pwrite(/x/f1, 2, "bar");
```

```
unlink(/x/log1);
```

Атомарная перезапись: v1

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, foo");
```

↑
— <offset, size, data>

```
pwrite(/x/f1, 2, "bar");
```

```
unlink(/x/log1);
```

Работает в режиме data-journal!

1.

/x/f1	a		f	o	o
/x/log1					

2.

/x/f1	a		f	o	o
/x/log1	2	3	f		

3.

/x/f1	a		b	o	o
/x/log1	2	3	f	o	o


Атомарная перезапись: v1

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, foo");
```

```
pwrite(/x/f1, 2, "bar");
```

```
unlink(/x/log1);
```



Работает в режиме data-journal!

Проблемы в режиме data-ordered!

Атомарная перезапись: v1

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, foo");
```

```
pwrite(/x/f1, 2, "bar");
```

```
unlink(/x/log1);
```



/x/f1	a		b	o	o
/x/log1					

Работает в режиме data-journal!

Проблемы в режиме data-ordered!

Атомарная перезапись: v2

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, foo");
```

```
fsync(/x/log1);
```

```
pwrite(/x/f1, 2, "bar");
```

```
fsync(/x/f1);
```

```
unlink(/x/log1);
```

Работает в режиме data-journal, data-ordered!

Атомарная перезапись: v2

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, foo");
```

```
fsync(/x/log1);
```

```
pwrite(/x/f1, 2, "bar");
```

```
fsync(/x/f1);
```

```
unlink(/x/log1);
```

Работает в режиме data-journal, data-ordered! **Не работает в режиме writeback!**

Атомарная перезапись: v2

```
creat(/x/log1);  
write(/x/log1, "2, 3, foo");  


---

fsync(/x/log1);
```

/x/f1	a		b	o	o
/x/log1	#	4	2	!	@

```
pwrite(/x/f1, 2, "bar");  
fsync(/x/f1);  
unlink(/x/log1);
```

Работает в режиме data-journal, data-ordered! **Не работает в режиме writeback!**

Атомарная перезапись: v3

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, checksum, foo");
```

```
fsync(/x/log1);
```

```
pwrite(/x/f1, 2, "bar");
```

```
fsync(/x/f1);
```

```
unlink(/x/log1);
```

Работает в режиме data-journal, data-ordered, writeback!

Атомарная перезапись: v3

```
creat(/x/log1);
```

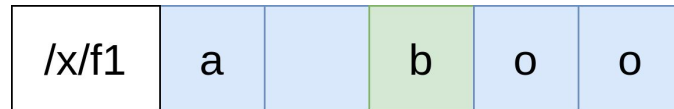
```
write(/x/log1, "2, 3, checksum, foo");
```

```
fsync(/x/log1);
```

```
pwrite(/x/f1, 2, "bar");
```

```
fsync(/x/f1);
```

```
unlink(/x/log1);
```



Работает в режиме data-journal, data-ordered, writeback! **man fsync наносит ответный удар**

Атомарная перезапись: v4

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, checksum, foo");
```

```
fsync(/x/log1);
```

```
fsync(/x);
```

```
pwrite(/x/f1, 2, "bar");
```

```
fsync(/x/f1);
```

```
unlink(/x/log1);
```

Calling **fsync()** does not necessarily ensure that the entry in the directory containing the file has also reached disk. For that an explicit **fsync()** on a file descriptor for the directory is also needed.

Работает в режиме data-journal, data-ordered, writeback!

Атомарная перезапись: v4

```
creat(/x/log1);
```

```
write(/x/log1, "2, 3, checksum, foo");
```

```
fsync(/x/log1);
```

```
fsync(/x);
```

```
pwrite(/x/f1, 2, "bar");
```

```
fsync(/x/f1);
```

```
unlink(/x/log1);
```

```
fsync(/x);
```

Работает в режиме data-journal, data-ordered, writeback!

Атомарная перезапись: Итоги

- Снова: ФС разные, опции разные, гарантии разные

Атомарная перезапись: Итоги

- Снова: ФС разные, опции разные, гарантии разные
- Оптимизация под конкретную ФС

Атомарная перезапись: Итоги

- Снова: ФС разные, опции разные, гарантии разные
- Оптимизация под конкретную ФС
- Буферизованный ввод вывод добавляет неопределенности

Другие неоправданные ожидания

Application	Types							Unique static vulnerabilities		
	Across-syscalls atomicity	Atomicity		Ordering		Durability				
		Appends and truncates Single-block overwrites Renames and unlinks		Safe file flush Safe renames Other		Safe file flush Other				
Leveldb1.10	1 ⁺	1	1	2	1	3	1	10		
Leveldb1.15	1	1	1	1	2			6		
LMDB		1						1		
GDBM	1		1		1		2	5		
HSQldb		1	2	1	3	2	1	10		
Sqlite-Roll							1	1		
Sqlite-WAL								0		
PostgreSQL		1						1		
Git	1		1	2	1	3	1	9		
Mercurial	2	1	1	1	4		2	10		
VMWare			1					1		
HDFS			1		1			2		
ZooKeeper			1		1	2		4		
Total	6	4	3	9	6	3	18	5	7	60

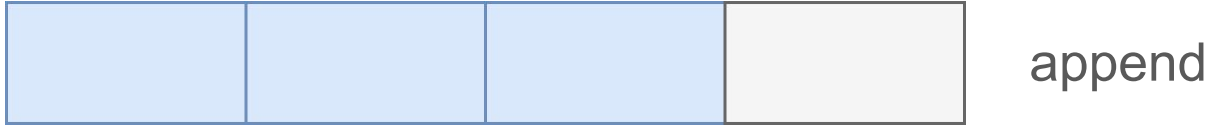
(a) Types.

Application	Silent errors	Data loss	Cannot open	Failed reads and writes	Other
Leveldb1.10	1	1	5	4	
Leveldb1.15	2		2	2	
LMDB					read-only open [†]
GDBM		2*	3*		
HSQldb	2	3	5		
Sqlite-Roll		1*			
Sqlite-WAL					
PostgreSQL			1 [†]		
Git	1*	3*	5*		3#*
Mercurial	2*	1*	6*	5	dirstate fail*
VMWare			1*		
HDFS			2*		
ZooKeeper	2*	2*			
Total	5	12	25	17	9

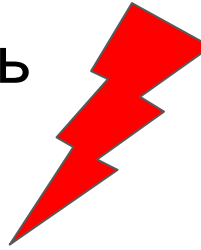
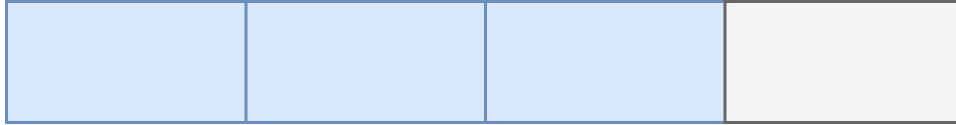
(b) Failure Consequences.

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

Восстановление после отказа: повреждение или частичная запись



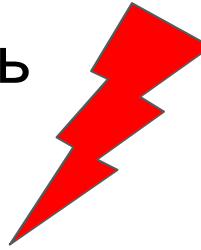
Восстановление после отказа:
повреждение или частичная запись



append

Восстановление после отказа: повреждение или частичная запись



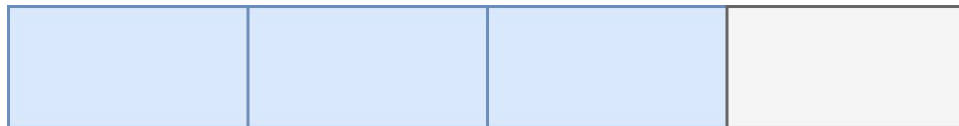
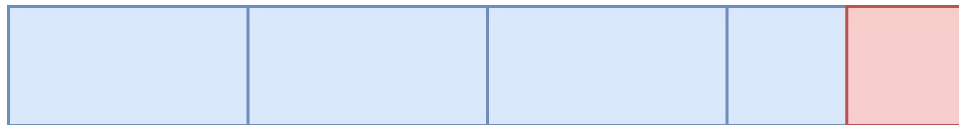

append



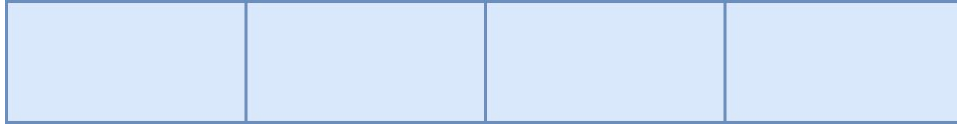
Восстановление после отказа: повреждение или частичная запись



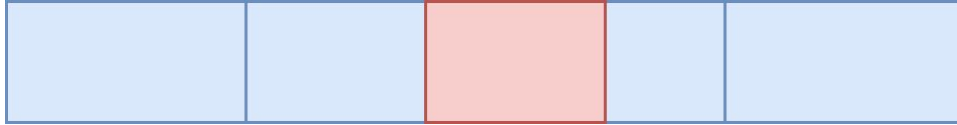

append



Восстановление после отказа: повреждение или частичная запись



Восстановление после отказа: повреждение или частичная запись



Что с этим всем делать?

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](#)

Что можно улучшить?

Можно использовать виртуалку

Что можно улучшить?

Можно использовать виртуалку

- `echo b > /proc/sysrq-trigger`

Will immediately reboot the system without syncing or unmounting your disks.

Что можно улучшить?

Можно использовать виртуалку

- `echo b > /proc/sysrq-trigger`
- LVM снапшоты

Что можно улучшить?

Можно использовать виртуалку

- `echo b > /proc/sysrq-trigger`
- LVM снапшоты

Плюсы:

- Относительная простота
инфраструктуры (для одного узла)
- Не нужно вносить изменения в код
приложения

Что можно улучшить?

Можно использовать виртуалку

- `echo b > /proc/sysrq-trigger`
- LVM снапшоты

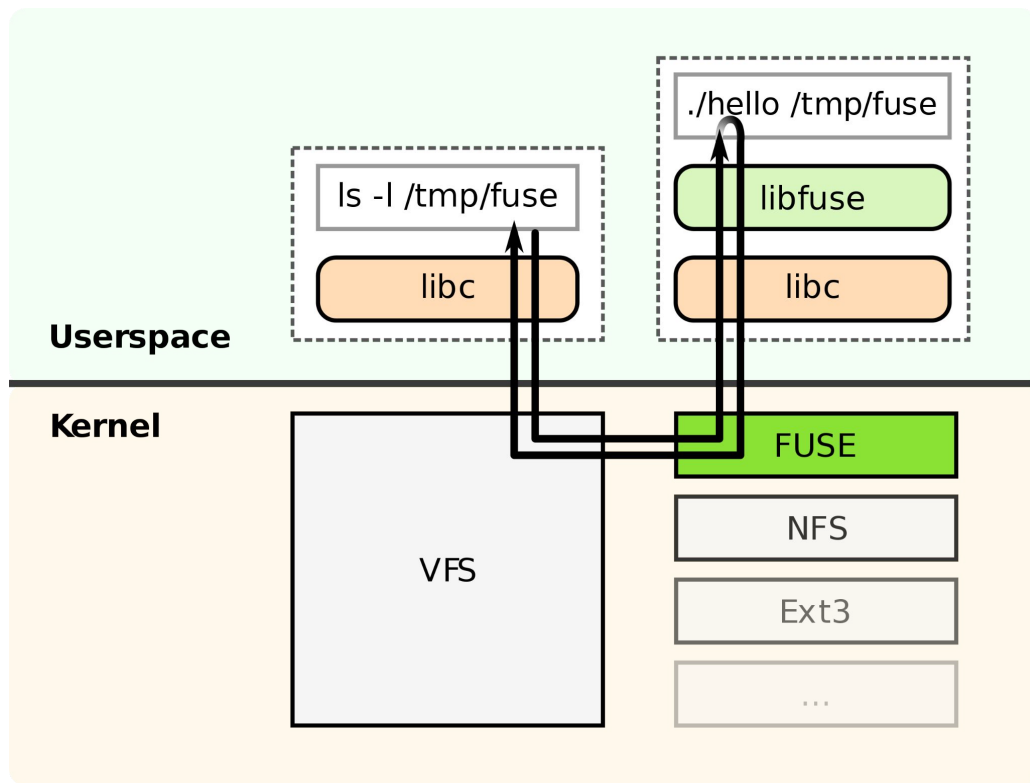
Плюсы:

- Относительная простота инфраструктуры (для одного узла)
- Не нужно вносить изменения в код приложения

Минусы:

- Воспроизводимость
- Проблемы с покрытием
- Удобство отладки

FUSE приходит на помощь



Интересные факты о FUSE

- Применяется в системах контроля версий ориентированных на монорепо (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](#)

Применение FUSE для тестирования

- Возврат ошибок в системных вызовах (read, write, fsync)
- Терять данные которые не были синхронизированы
- Частично записывать данные

Ссылки:

- [unreliablefs](#)
- [lazyfs](#)
- jepsen [Request for a Lazy Filesystem](#)

Применение FUSE для тестирования

- Удобнее чем виртуалки
- Внедрение ошибок на конкретные операции вместо случайных падений
- Не нужно изменять код приложения
- Воспроизводимость получше, но все равно не гарантируется
- ~Платформозависимость

ALICE

ALICE

Workload:

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

ALICE

Workload:

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


Checker:

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

ALICE

Workload:

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



strace

Checker:

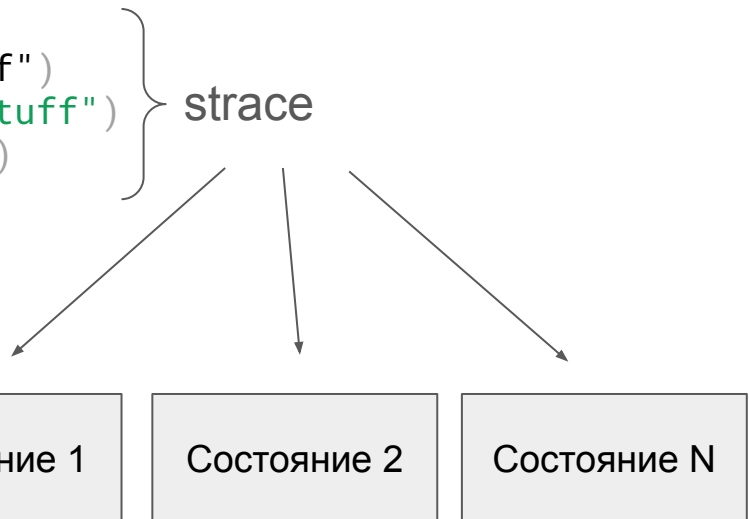
```
if output.contains("ok"):  
    f = open("f")  
    assert f.contains("stuff")
```

ALICE

Workload:

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

} strace



```
graph TD  
    A["f = open('f')  
f.write('stuff')  
print('ok')"] --- B["}] strace"]  
B --> C["Состояние 1"]  
B --> D["Состояние 2"]  
B --> E["Состояние N"]
```

Checker:

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

ALICE

Workload:

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

} strace

The diagram shows the workload code on the left, followed by a right curly brace and the word 'strace'. Three arrows originate from the bottom of the 'strace' label and point to three separate boxes below. The boxes are labeled 'Состояние 1', 'Состояние 2', and 'Состояние N'.

Состояние 1

Состояние 2

Состояние N

Checker:

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

The diagram shows the checker code on the right. A curved arrow originates from the bottom of the 'Состояние N' box and points up towards the checker code.

ALICE

- Лучшее покрытие

ALICE

- Лучшее покрытие
- Код менять не нужно

ALICE

- Лучшее покрытие
- Код менять не нужно
- Исходники опубликованы

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы
- Не все возможности описанные в статье есть в опубликованной версии

ALICE

- Лучше покрытие
- Код менять не нужно
- Исходники опубликованы
- Не все возможности описанные в статье есть в опубликованной версии
- Проект не поддерживается

ALICE

- Лучше покрытие
 - Код менять не нужно
 - Исходники опубликованы
- Не все возможности описанные в статье есть в опубликованной версии
 - Проект не поддерживается
 - Использование `strace` не совместимо с `io_uring`

ALICE

- Лучшее покрытие
- Код менять не нужно
- Исходники опубликованы
- Не все возможности описанные в статье есть в опубликованной версии
- Проект не поддерживается
- Использование strace не совместимо с io_uring

Итог: можно ограниченно использовать

Путь в светлое будущее и другие исследования по теме

Путь в светлое будущее и другие исследования по теме

- Применение опыта верификации моделей памяти

Путь в светлое будущее и другие исследования по теме

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

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.

Итог:

- Все сломано

Итог:

- Все сломано
- С этим можно жить

Итог:

- Все сломано
- С этим можно жить
- Не изобретать велосипеды, использовать базы данных

Итог:

- Все сломано
- С этим можно жить
- Не изобретать велосипеды, использовать базы данных
- Если изобрести велосипед все таки нужно, использовать правильные инструменты

Итог:

- Все сломано
- С этим можно жить
- Не изобретать велосипеды, использовать базы данных
- Если изобрести велосипед все таки нужно, использовать правильные инструменты
- Делать бекапы

Со смертью этого персонажа нить вашей судьбы обрывается. Загрузите сохраненную игру, чтобы восстановить течение судьбы, или живите дальше в проклятом мире, который сами и создали.

СПАСИБО ЗА ВНИМАНИЕ

[@LizardWizzard](https://github.com/LizardWizzard) tg/github

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](#)
- [Can Applications Recover from fsync Failures?](#)
- [POSIX fsync](#)
- [POSIX v. reality: A position on O_PONIES](#)

Библиография: 2

- [\[PATCH\] 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](#)
- [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](#)

Библиография: 3

- [RFUSE: Modernizing Userspace Filesystem Framework through Scalable Kernel-Userspace Communication](#)
- [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](#)

