Betriebssysteme

13. Tutorium - Files Systems

Peter Bohner

6. Februar 2025

ITEC - Operating Systems Group

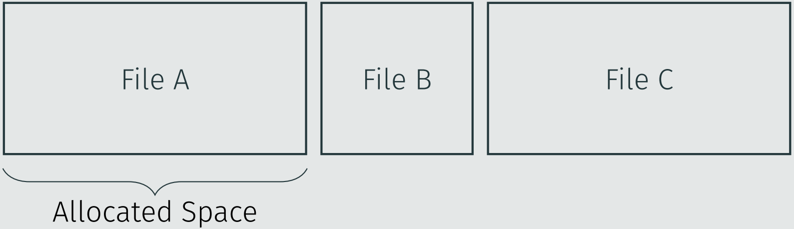
Disk Space Allocation

Contiguous Allocation

What is that?

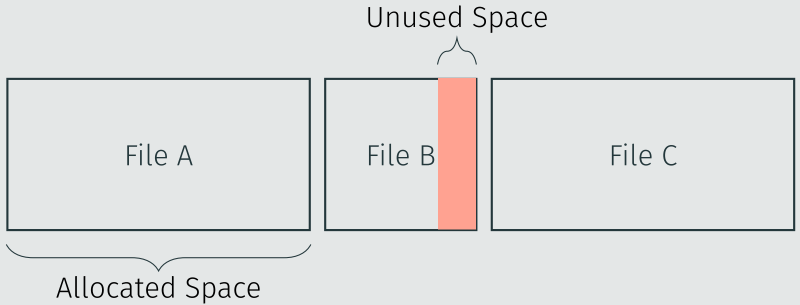
Contiguous Allocation

What is that?



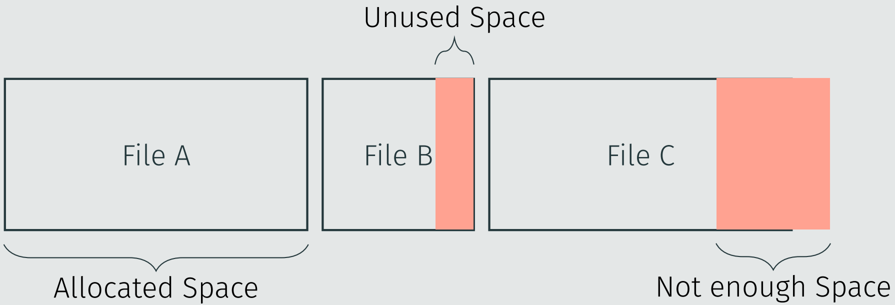
Contiguous Allocation

What is that?



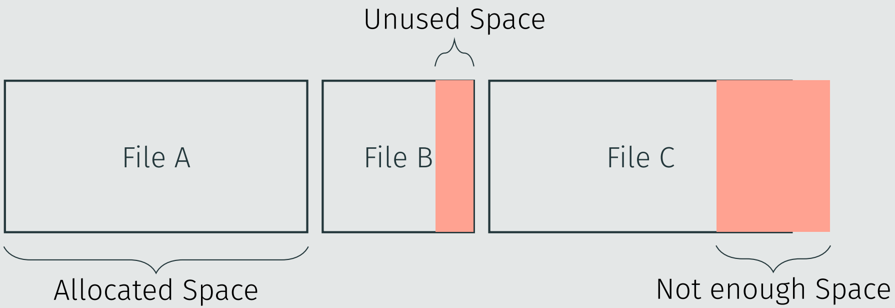
Contiguous Allocation

What is that?



Contiguous Allocation

What is that?



Challenges

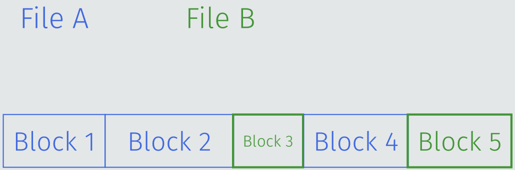
- Sizing your block (internal fragmentation, growth)
- External fragmentation

Chained Allocation

What's that?

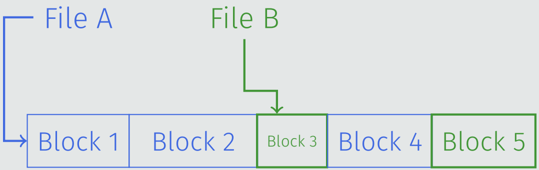
Chained Allocation

What’s that?



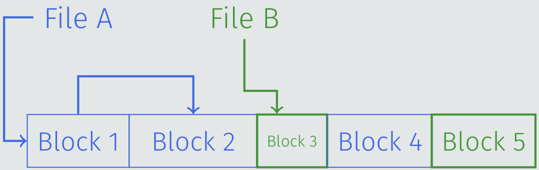
Chained Allocation

What's that?



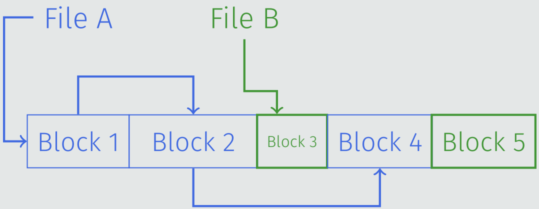
Chained Allocation

What's that?



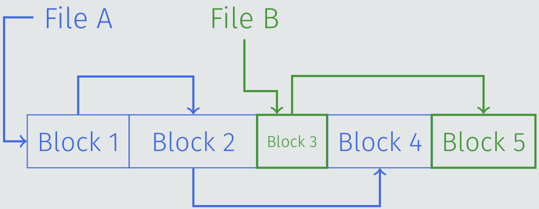
Chained Allocation

What's that?



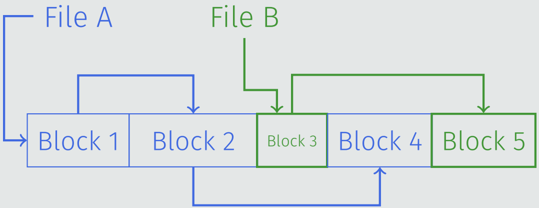
Chained Allocation

What's that?



Chained Allocation

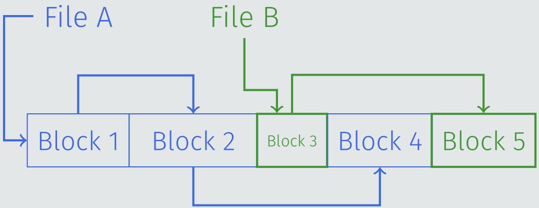
What's that?



Benefits? Drawbacks?

Chained Allocation

What's that?

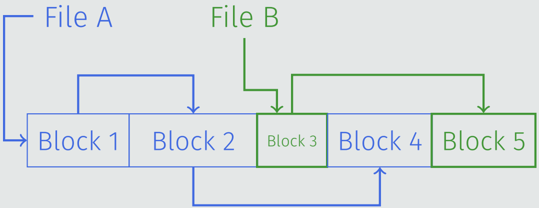


Benefits? Drawbacks?

+ No longer need a contiguous chunk

Chained Allocation

What's that?

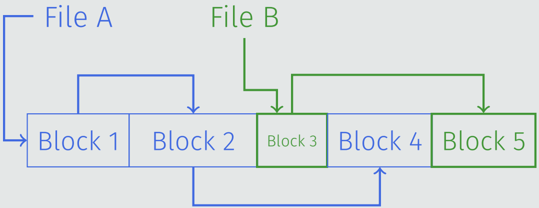


Benefits? Drawbacks?

+ No longer need a contiguous chunk
- Only sequential access

Chained Allocation

What's that?

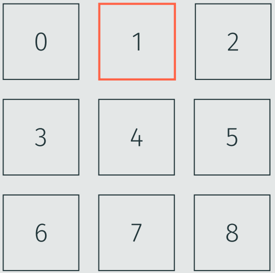


Benefits? Drawbacks?

- \+ No longer need a contiguous chunk
- – Only sequential access
- – A single corrupted pointer is very bad news

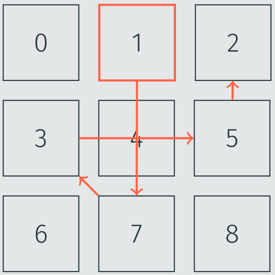
Linked List Allocation With FAT

What is that?



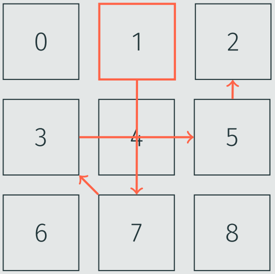
Linked List Allocation With FAT

What is that?



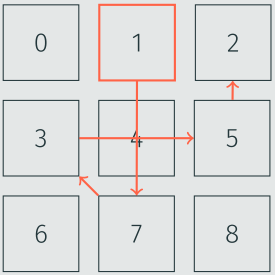
Linked List Allocation With FAT

What is that?

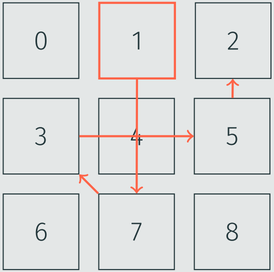


Linked List Allocation With FAT

What is that?



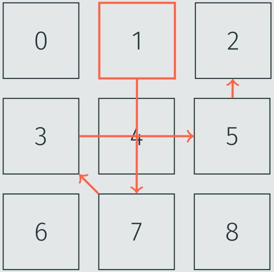
What is that?



0		
1	7	⇐ File A
2		
3	5	
4		
5		
6		
7	3	
8		

Linked List Allocation With FAT

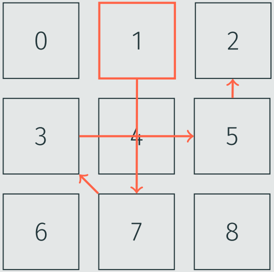
What is that?



0	
1	7 ⇐ File A
2	
3	5
4	
5	2
6	
7	3
8	

Linked List Allocation With FAT

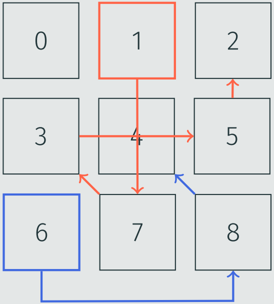
What is that?



0		
1	7	⇐ File A
2	-1	
3	5	
4		
5	2	
6		
7	3	
8		

Linked List Allocation With FAT

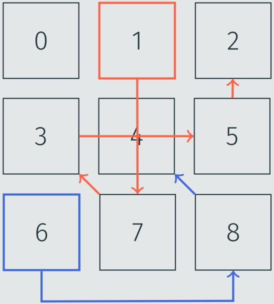
What is that?



0		
1	7	⇐ File A
2	-1	
3	5	
4		
5	2	
6		
7	3	
8		

Linked List Allocation With FAT

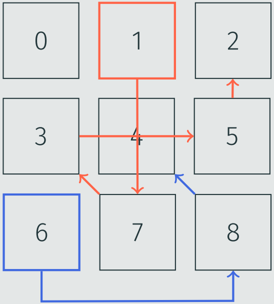
What is that?



0		
1	7	⇐ File A
2	-1	
3	5	
4		
5	2	
6	8	⇐ File B
7	3	
8		

Linked List Allocation With FAT

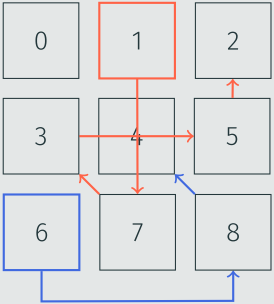
What is that?



0		
1	7	⇐ File A
2	-1	
3	5	
4		
5	2	
6	8	⇐ File B
7	3	
8	4	

Linked List Allocation With FAT

What is that?



0		
1	7	⇐ File A
2	-1	
3	5	
4	-1	
5	2	
6	8	⇐ File B
7	3	
8	4	

Linked List Allocation With FAT

Benefits? Drawbacks?

Linked List Allocation With FAT

Benefits? Drawbacks?

+ Hopefully fits in RAM $\Rightarrow$ Iteration fast

Linked List Allocation With FAT

Benefits? Drawbacks?

+ Hopefully fits in RAM $\Rightarrow$ Iteration fast
+ Even if it doesn't fit: Less disk seeks

Linked List Allocation With FAT

Benefits? Drawbacks?

- + Hopefully fits in RAM $\Rightarrow$ Iteration fast
- + Even if it doesn't fit: Less disk seeks
- – Size depends on size of hard disk (one entry per Block)

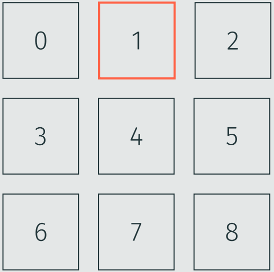
Linked List Allocation With FAT

Benefits? Drawbacks?

- \+ Hopefully fits in RAM $\Rightarrow$ Iteration fast
- \+ Even if it doesn't fit: Less disk seeks
- \- Size depends on size of hard disk (one entry per Block)
- \- Might be too large to be cached in RAM (on large disk)

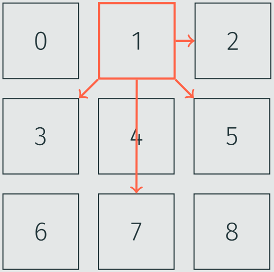
Indexed Allocation

What is that?



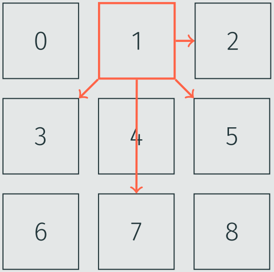
Indexed Allocation

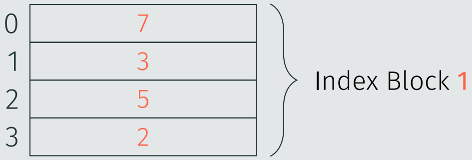
What is that?



Indexed Allocation

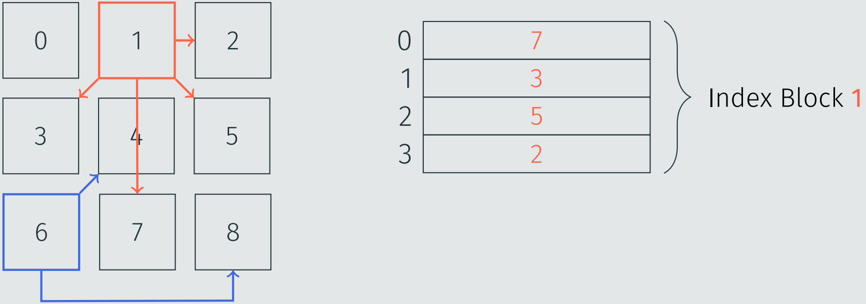
What is that?





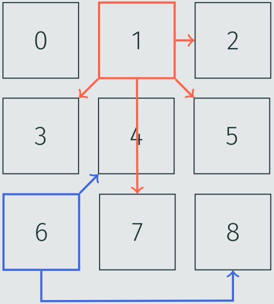
Indexed Allocation

What is that?



Indexed Allocation

What is that?



0	7
1	3
2	5
3	2

Index Block 1

0	8
1	4

Index Block 6

Indexed Allocation

Benefits? Drawbacks?

Indexed Allocation

Benefits? Drawbacks?

+ Hopefully fits in RAM $\Rightarrow$ Lookups fast

Indexed Allocation

Benefits? Drawbacks?

- \+ Hopefully fits in RAM $\Rightarrow$ Lookups fast
- \+ Random access possible (just fetch the ith block)

Indexed Allocation

Benefits? Drawbacks?

- \+ Hopefully fits in RAM $\Rightarrow$ Lookups fast
- \+ Random access possible (just fetch the ith block)
- \- File size limited by how many references fit into one block

Indexed Allocation

Benefits? Drawbacks?

- \+ Hopefully fits in RAM $\Rightarrow$ Lookups fast
- \+ Random access possible (just fetch the ith block)
- – File size limited by how many references fit into one block
- – Wasteful for small files

Indexed Allocation

Benefits? Drawbacks?

- + Hopefully fits in RAM $\Rightarrow$ Lookups fast
- + Random access possible (just fetch the ith block)
- – File size limited by how many references fit into one block
- – Wasteful for small files

How could you store *Huge* files?

Indexed Allocation

Benefits? Drawbacks?

- \+ Hopefully fits in RAM ⇒ Lookups fast
- \+ Random access possible (just fetch the *i*th block)
- – File size limited by how many references fit into one block
- – Wasteful for small files

How could you store *Huge* files?

- – Increase the Block size

Indexed Allocation

Benefits? Drawbacks?

\+ Hopefully fits in RAM $\Rightarrow$ Lookups fast

\+ Random access possible (just fetch the *i*th block)

\- File size limited by how many references fit into one block

\- Wasteful for small files

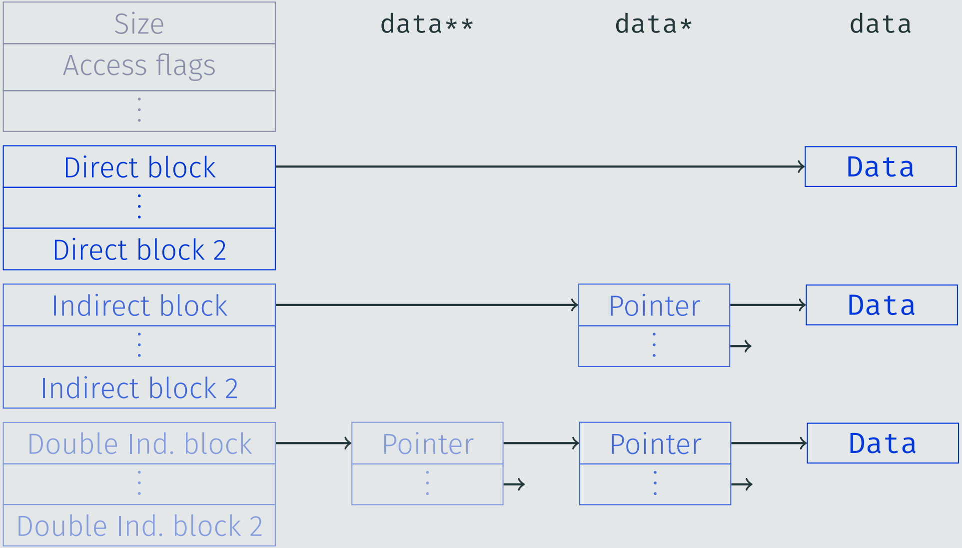
How could you store *Huge* files?

\- Increase the Block size

\+ Indirection! Make the index block link to other index blocks (like pagetables)
You can also mix that: First *N* pointers point to data blocks, next to indirect blocks, next to double-indirect blocks, etc.

Indexed Allocations With Inodes

What does an inode look like?



Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Calculating the amount of pointers to blocks

Pointers per Block: 8 KiB / 4 $bytes = 2^{13} / 2^{2} = 2^{11} = 2048$

Direct pointers

Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Calculating the amount of pointers to blocks

Pointers per Block: 8 KiB / 4 $bytes = 2^{13} / 2^2 = 2^{11} = 2048$

Direct pointers	12 pointers to blocks
One single-indirect block	

Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Calculating the amount of pointers to blocks

Pointers per Block: 8 KiB / 4 $bytes = 2^{13} / 2^{2} = 2^{11} = 2048$

Direct pointers	12 pointers to blocks
One single-indirect block	2048 pointers to blocks
One double-indirect block	

Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Calculating the amount of pointers to blocks

Pointers per Block: 8 KiB / 4 $bytes = 2^{13} / 2^{2} = 2^{11} = 2048$

Direct pointers	12 pointers to blocks
One single-indirect block	2048 pointers to blocks
One double-indirect block	$2048 \cdot 2048$ pointers to blocks
One triple-indirect block	

Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Calculating the amount of pointers to blocks

Pointers per Block: 8 KiB / 4 $bytes = 2^{13} / 2^{2} = 2^{11} = 2048$

Direct pointers	12 pointers to blocks
One single-indirect block	2048 pointers to blocks
One double-indirect block	$2048 \cdot 2048$ pointers to blocks
One triple-indirect block	$2048 \cdot 2048 \cdot 2048$ pointers to blocks

Indexed Allocations With Inodes

What is the maximum file size?

Assume that disk blocks are 8 KiB in size and a pointer to a disk block is 4 bytes long. An inode contains 12 pointers to direct blocks, and one pointer to a single, double, and triple indirect block, respectively.

Calculating the amount of pointers to blocks

Pointers per Block: 8 KiB / 4 $bytes = 2^{13} \ / \ 2^2 = 2^{11} = 2048$

Direct pointers	12 pointers to blocks
One single-indirect block	2048 pointers to blocks
One double-indirect block	$2048 \cdot 2048$ pointers to blocks
One triple-indirect block	$2048 \cdot 2048 \cdot 2048$ pointers to blocks

Final result

$(12 + 2048 + 2048^2 + 2048^3) \cdot 8KiB \approx 64TiB$

File System Implementation

Hard links

Hard links

Pointer to the same inode.

$\Rightarrow$ *Everything* is identical: Size, content, access time, mode, ...

Can therefore *not* cross file system boundaries.

Hard links

Pointer to the same inode.

$\Rightarrow$ *Everything* is identical: Size, content, access time, mode, …

Can therefore *not* cross file system boundaries.

Symlinks

Hard links

Pointer to the same inode.

$\Rightarrow$ *Everything* is identical: Size, content, access time, mode, ...

Can therefore *not* cross file system boundaries.

Symlinks

Are (mostly) normal files containing a filepath with their own access time, mode, ... When programs access the file the OS transparently:

1. Reads the contents of the symlink file
2. Resolves the file path it read
3. Performs the operation on that path instead

Can cross file system boundaries or point to non-existent files or change what they point to if the file is moved, ...

Having Fun With Paths I

Hard and symlink renames

```
echo "Hello" > test.txt
ln test.txt hardlink.txt
ln -s test.txt symlink.txt

mv test.txt renamed_test.txt
// Is the hardlink / symlink broken?
```

Having Fun With Paths I

Hard and symlink renames

```
echo "Hello" > test.txt
ln test.txt hardlink.txt
ln -s test.txt symlink.txt

mv test.txt renamed_test.txt
// Is the hardlink / symlink broken?
```

Reaching for the stars

Does this work (on my machine ;)?

```
cd /tmp/
echo "Hello" > test.txt
ln test.txt ~/test_link.txt
```

Having Fun With Paths I

Hard and symlink renames

```
echo "Hello" > test.txt
ln test.txt hardlink.txt
ln -s test.txt symlink.txt

mv test.txt renamed_test.txt
// Is the hardlink / symlink broken?
```

Reaching for the stars

Does this work (on my machine ;)?

```
cd /tmp/
echo "Hello" > test.txt
ln test.txt ~/test_link.txt
```

No, as `/tmp` is mounted as tempfs and `/` as ext4.

Having Fun With Paths II

Hard and symlink renames

```
echo "Hello" > test.txt
ln test.txt hardlink.txt
ln -s test.txt symlink.txt

cp test.txt renamed_test.txt
rm test.txt
// Is the hardlink / symlink broken?
```

Having Fun With Paths II

Hard and symlink renames

```
echo "Hello" > test.txt
ln test.txt hardlink.txt
ln -s test.txt symlink.txt

cp test.txt renamed_test.txt
rm test.txt
// Is the hardlink / symlink broken?

echo "Hello" > test.txt
// Is the hardlink / symlink still broken?
```

Having Fun With Paths II

Hard and symlink renames

```
echo "Hello" > test.txt
ln test.txt hardlink.txt
ln -s test.txt symlink.txt

cp test.txt renamed_test.txt
rm test.txt
// Is the hardlink / symlink broken?

echo "Hello" > test.txt
// Is the hardlink / symlink still broken?
```

If you delete a hardlink, how can the FS know when it can delete the file?

If you delete a hardlink, how can the FS know when it can delete the file?

If the refcount stored in the inode is zero

Misc

If you delete a hardlink, how can the FS know when it can delete the file?

If the refcount stored in the inode is zero

How can directories be implemented? What information is stored in them?

Misc

If you delete a hardlink, how can the FS know when it can delete the file?

If the refcount stored in the inode is zero

How can directories be implemented? What information is stored in them?

As a normal file with variable-length entries consisting of

- The filename
- The inode that represents that file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Inodes

Which of the following data are typically stored in an inode?

1. filename
2. name of containing directory
3. file size
4. file type
5. number of symlinks to file
6. name/location of symlinks
7. number of hardlinks
8. name/location of hardlinks
9. access rights
10. timestamps (last access/modify)
11. file contents
12. ordered list of blocks occupied by file

Virtual File System

Virtual File System

What is the purpose of the VFS layer in an OS?

Virtual File System

What is the purpose of the VFS layer in an OS?

Abstraction. Provide a high-level API that works no matter what file system you use (XFS, ext4, NTFS, SMB, IliasFUSE, ...).

What is the purpose of the VFS layer in an OS?

Abstraction. Provide a high-level API that works no matter what file system you use (XFS, ext4, NTFS, SMB, IliasFUSE, …).

What are some drawbacks / problems?

Virtual File System

What is the purpose of the VFS layer in an OS?

Abstraction. Provide a high-level API that works no matter what file system you use (XFS, ext4, NTFS, SMB, IliasFUSE, ...).

What are some drawbacks / problems?

- Lowest common denominator: Might hide special FS features

Virtual File System

What is the purpose of the VFS layer in an OS?

Abstraction. Provide a high-level API that works no matter what file system you use (XFS, ext4, NTFS, SMB, IliasFUSE, ...).

What are some drawbacks / problems?

- Lowest common denominator: Might hide special FS features

$\Rightarrow$ Allow accessing underlying FS (e.g. via `ioctl`)

$\Rightarrow$ Lose compatibility with other file systems

Virtual File System

What is the purpose of the VFS layer in an OS?

Abstraction. Provide a high-level API that works no matter what file system you use (XFS, ext4, NTFS, SMB, IliasFUSE, ...).

What are some drawbacks / problems?

- Lowest common denominator: Might hide special FS features

$\Rightarrow$ Allow accessing underlying FS (e.g. via `ioctl`)

$\Rightarrow$ Lose compatibility with other file systems

What happens when you mount a filesystem?

E.g. `mount /dev/sda1 /mnt`.

Virtual File System

What is the purpose of the VFS layer in an OS?

Abstraction. Provide a high-level API that works no matter what file system you use (XFS, ext4, NTFS, SMB, IliasFUSE, ...).

What are some drawbacks / problems?

- Lowest common denominator: Might hide special FS features

$\Rightarrow$ Allow accessing underlying FS (e.g. via `ioctl`)

$\Rightarrow$ Lose compatibility with other file systems

What happens when you mount a filesystem?

E.g. `mount /dev/sda1 /mnt`.

Makes files from that file system accessible. It will make the given path (e.g. `/mnt`) the *root* of the new file system. Any access to `/tmp/*` is stripped of the `/tmp/` prefix and then searched in the mounted file system.

FUSE

File System in **USE**rspace.

- Allows users to write their own file system without writing kernel code

FUSE

File System in **USE**rspace.

- Allows users to write their own file system without writing kernel code
- Runs as an unprivileged user process

FUSE

FUSE

File System in USErspace.

- Allows users to write their own file system without writing kernel code
- Runs as an unprivileged user process
- Is really awsome! You suddenly can use all your normal tools on whatever the FUSE filesystem exposes!

File System Cache

File System Cache

Is data persisted after a call to `write`? If not, why and when?

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first
- Even after that, it doesn't directly get written to disk. Enter...

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first
- Even after that, it doesn't directly get written to disk. Enter...The *File System Cache*!

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first
- Even after that, it doesn't directly get written to disk. Enter...The *File System Cache*!
- Why? Writing to disk every time is *reeeaaaally slow*, so it is probably cached in memory and flushed in chunks

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first
- Even after that, it doesn’t directly get written to disk. Enter...The *File System Cache*!
- Why? Writing to disk every time is *reeeaaaally slow*, so it is probably cached in memory and flushed in chunks

The standard question: Why is a cache even helpful?

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first
- Even after that, it doesn't directly get written to disk. Enter...The *File System Cache*!
- Why? Writing to disk every time is *reeeaaaally slow*, so it is probably cached in memory and flushed in chunks

The standard question: Why is a cache even helpful?

- Temporal and spatial locality

File System Cache

Is data persisted after a call to `write`? If not, why and when?

- Data is probably buffered by libc first
- Even after that, it doesn't directly get written to disk. Enter...The *File System Cache*!
- Why? Writing to disk every time is *reeeaaaally slow*, so it is probably cached in memory and flushed in chunks

The standard question: Why is a cache even helpful?

- Temporal and spatial locality
- When exactly is it flushed to disk then? Is that even important to know/control?

File System Cache

Is data persisted after a call to `write`? If not, why and when?

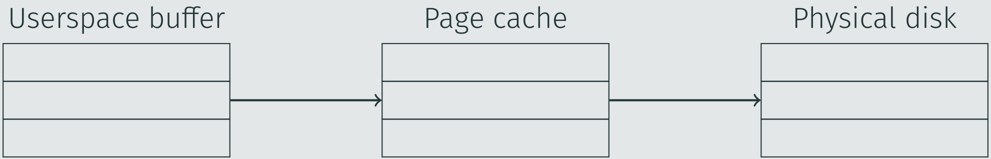
- Data is probably buffered by libc first
- Even after that, it doesn't directly get written to disk. Enter...The *File System Cache*!
- Why? Writing to disk every time is *reeeaaaally slow*, so it is probably cached in memory and flushed in chunks

The standard question: Why is a cache even helpful?

- Temporal and spatial locality
- When exactly is it flushed to disk then? Is that even important to know/control?
- Using `flush` (for buffers) and `fsync` (for the page cache) and after some time by a daemon

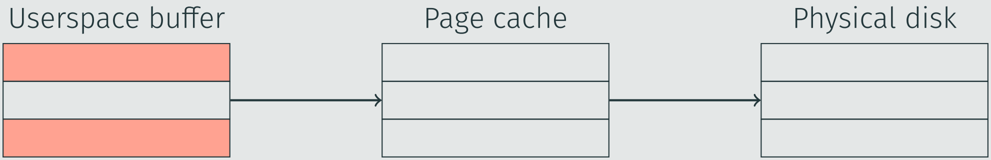
Page cache interactions

Let's talk about the page cache



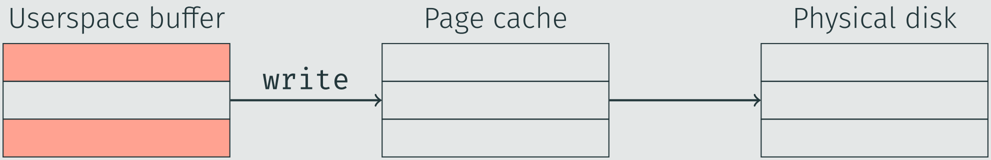
Page cache interactions

Let's talk about the page cache



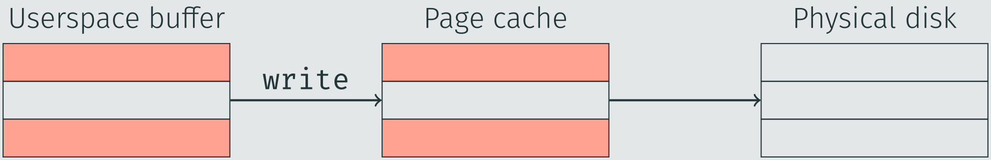
Page cache interactions

Let's talk about the page cache



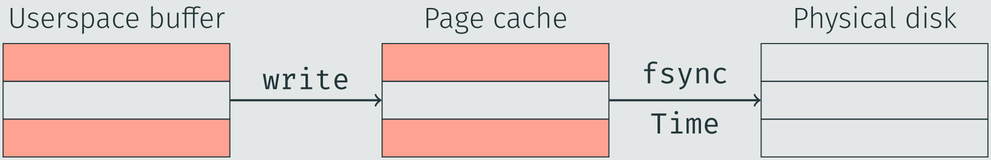
Page cache interactions

Let's talk about the page cache



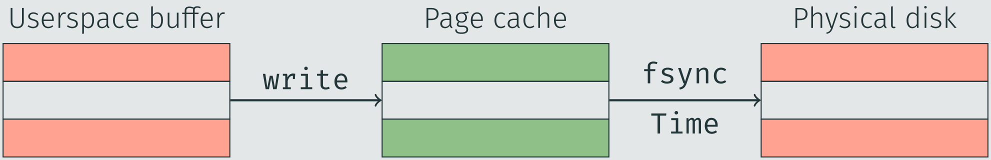
Page cache interactions

Let's talk about the page cache



Page cache interactions

Let’s talk about the page cache



Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Fix Easy to implement

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Fix Easy to implement

Fix Can give hard guarantees (real-time systems anyone?)

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

- Fix Easy to implement
- Fix Can give hard guarantees (real-time systems anyone?)
- Fix Can't adapt to different workloads

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

- Fix Easy to implement
- Fix Can give hard guarantees (real-time systems anyone?)
- Fix Can't adapt to different workloads

When do you load data in the page cache?

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Fix Easy to implement

Fix Can give hard guarantees (real-time systems anyone?)

Fix Can't adapt to different workloads

When do you load data in the page cache?

Read-ahead: Read more data in the cache than requested

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Fix Easy to implement

Fix Can give hard guarantees (real-time systems anyone?)

Fix Can't adapt to different workloads

When do you load data in the page cache?

Read-ahead: Read more data in the cache than requested

+ Good sequential performance

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Fix Easy to implement

Fix Can give hard guarantees (real-time systems anyone?)

Fix Can't adapt to different workloads

When do you load data in the page cache?

Read-ahead: Read more data in the cache than requested

- + Good sequential performance
- + Improved throughput if files are sequential

Sizing the page cache

How large would you make it? Do you give it a fixed size or let it grow?

Fix Easy to implement

Fix Can give hard guarantees (real-time systems anyone?)

Fix Can't adapt to different workloads

When do you load data in the page cache?

Read-ahead: Read more data in the cache than requested

- + Good sequential performance
- + Improved throughput if files are sequential
- - Wasted time and memory if not needed

The File System Cache and mmap

How could you implement `mmap` with the file system cache?

The File System Cache and mmap

How could you implement `mmap` with the file system cache?

Just map the file system cache in the process's memory (and handle faults!)

$\Rightarrow$ Acts as a shared memory segment

You synchronize accesses yourself :(

The File System Cache and mmap

How could you implement `mmap` with the file system cache?

Just map the file system cache in the process's memory (and handle faults!)

$\Rightarrow$ Acts as a shared memory segment

You synchronize accesses yourself :(

`read()` and `write()` and the file system cache

The File System Cache and mmap

How could you implement `mmap` with the file system cache?

Just map the file system cache in the process's memory (and handle faults!)

$\Rightarrow$ Acts as a shared memory segment

You synchronize accesses yourself :(

`read()` and `write()` and the file system cache

- `read()`: Copy data from cache to your application buffers

The File System Cache and mmap

How could you implement `mmap` with the file system cache?

Just map the file system cache in the process's memory (and handle faults!)

$\Rightarrow$ Acts as a shared memory segment

You synchronize accesses yourself :(

`read()` and `write()` and the file system cache

- `read()`: Copy data from cache to your application buffers
- `write()`: Copy data from your application buffers to the file system cache

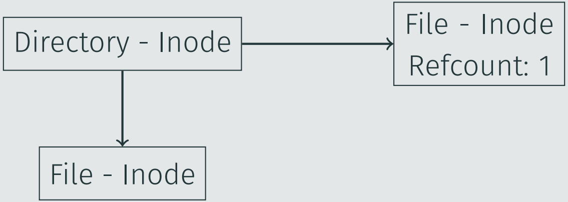
File system synchronizes access!

Modern File Systems

Crashing Is Fun

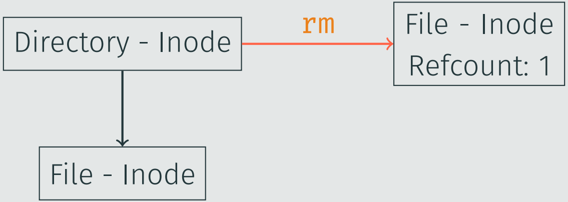
Your computer crashes (e.g. your power fails). What horrible death does your file system die?

A fun game for the whole family and your „why would I need backups“ crowd



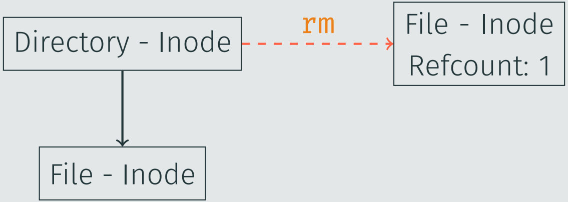
Your computer crashes (e.g. your power fails). What horrible death does your file system die?

A fun game for the whole family and your „why would I need backups“ crowd



Your computer crashes (e.g. your power fails). What horrible death does your file system die?

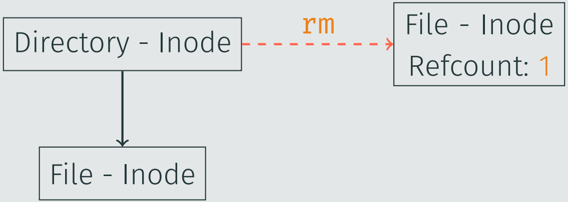
A fun game for the whole family and your „why would I need backups“ crowd



Crashing Is Fun

Your computer crashes (e.g. your power fails). What horrible death does your file system die?

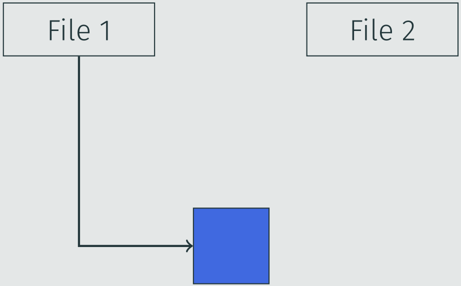
A fun game for the whole family and your „why would I need backups“ crowd



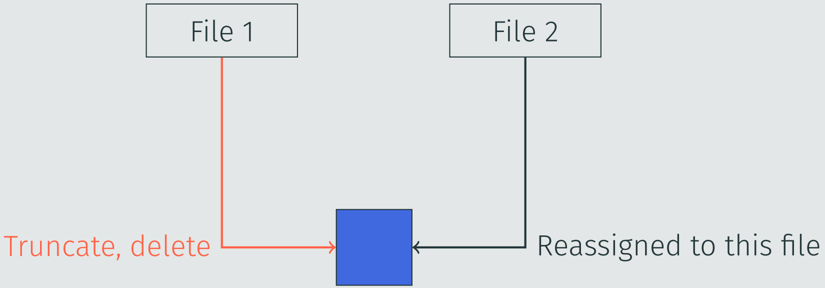
The inode has a refcount of 1 but is no longer referenced!

Crashing Is Fun

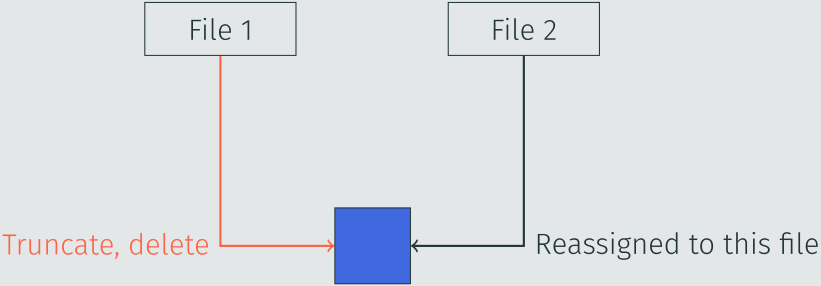
Where is Data?



Where is Data?

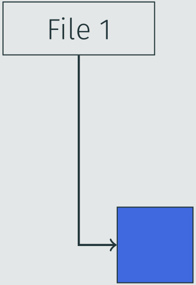


Where is Data?

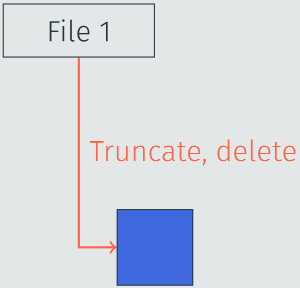


Both files might point to this block!

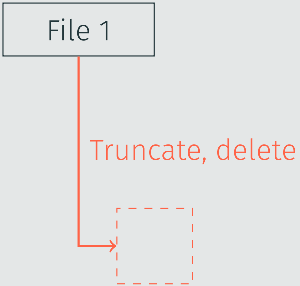
Where is Data?



Where is Data?

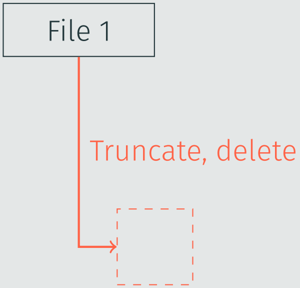


Where is Data?



Block still referenced but marked free

Where is Data?



Block still referenced but marked free

Or multiple directory entries with the same name, as another problem

Ordering

What might be the outcome?

```
// create a file
echo "Hey" > a.txt
// And a second
echo "Hello" > b.txt
// CRASH
```

Ordering

What might be the outcome?

```
// create a file
echo "Hey" > a.txt
// And a second
echo "Hello" > b.txt
// CRASH
```

- Both files exist

Ordering

What might be the outcome?

```
// create a file
echo "Hey" > a.txt
// And a second
echo "Hello" > b.txt
// CRASH
```

- Both files exist
- No files exist

Ordering

What might be the outcome?

```
// create a file
echo "Hey" > a.txt
// And a second
echo "Hello" > b.txt
// CRASH
```

- Both files exist
- No files exist
- Only `a` exists

Ordering

What might be the outcome?

```
// create a file
echo "Hey" > a.txt
// And a second
echo "Hello" > b.txt
// CRASH
```

- Both files exist
- No files exist
- Only a exists
- Only b exists !!

Reordering is *allowed* (unless you take precautions)

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

Error Detection

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode

Error Detection

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode
- Do all directories contain valid entries (and just one with a given name)

Error Detection

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode
- Do all directories contain valid entries (and just one with a given name)
- Is every directory / file connected to the directory tree

Error Detection

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode
- Do all directories contain valid entries (and just one with a given name)
- Is every directory / file connected to the directory tree
- Is the refcount consistent with the number of hardlinks? What do you do if not?
  $refcount > 0$ but no links?

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode
- Do all directories contain valid entries (and just one with a given name)
- Is every directory / file connected to the directory tree
- Is the refcount consistent with the number of hardlinks? What do you do if not?
  $refcount > 0$ but no links? $\Rightarrow$ Attach to `lost+found`
  Update the refcount to be consistent

How could you detect those inconsistencies?

Use the **f**sck (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode
- Do all directories contain valid entries (and just one with a given name)
- Is every directory / file connected to the directory tree
- Is the refcount consistent with the number of hardlinks? What do you do if not?
  $refcount > 0$ but no links? $\Rightarrow$ Attach to `lost+found`
  Update the refcount to be consistent

Can this fix your application data?

Error Detection

How could you detect those inconsistencies?

Use the **fsck** (**f**ile **s**ystem **c**onsistency chec**k**) program. And what could that do?

- Check whether all data blocks are referenced by *exactly one* inode
- Do all directories contain valid entries (and just one with a given name)
- Is every directory / file connected to the directory tree
- Is the refcount consistent with the number of hardlinks? What do you do if not?
  refcount > 0 but no links? $\Rightarrow$ Attach to `lost+found`
  Update the refcount to be consistent

Can this fix your application data?

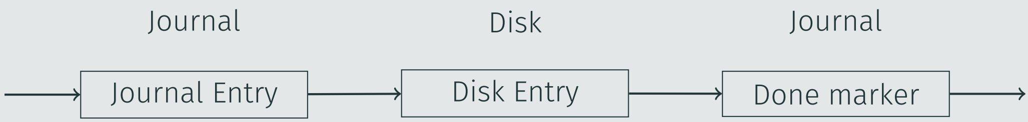
No! It just tries to keep the filesystem internally consistent, it won't find corrupted data blocks

Journal File System - Recovery

General principle

Walk over the journal and execute any outstanding entries.

Let's crash



What to do

The happy path, everything's nice

General principle

Walk over the journal and execute any outstanding entries.

Let’s crash

What to do

$\Rightarrow$ We didn’t write anything!

$\Rightarrow$ Operation failed

General principle

Walk over the journal and execute any outstanding entries.

Let's crash

What to do

⇒ Invalid checksum

⇒ Skip entry

⇒ Operation failed

Journal File System - Recovery

General principle

Walk over the journal and execute any outstanding entries.

Let's crash

What to do

⇒ Non-terminated journal entry

⇒ Retry and complete operation

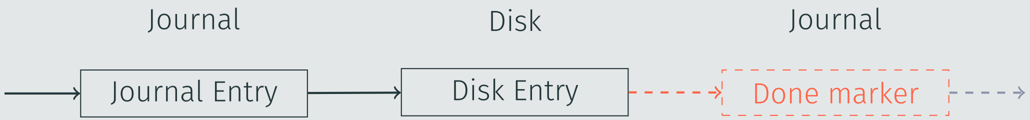
⇒ Operation successful

Journal File System - Recovery

General principle

Walk over the journal and execute any outstanding entries.

Let’s crash



What to do

⇒ Consistent state

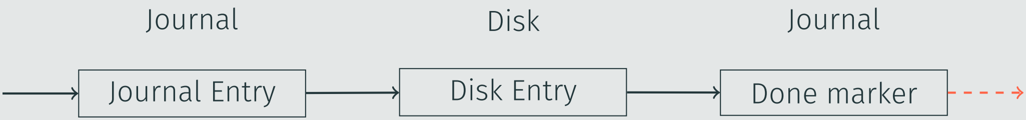
⇒ Execute operation again

⇒ Operation successful

General principle

Walk over the journal and execute any outstanding entries.

Let’s crash



What to do

⇒ Consistent state

⇒ Don’t do anything

⇒ Operation successful

Physical vs logical logging

What data could you log in the journal?

Journal

Physical vs logical logging

What data could you log in the journal?

- **Logical logging**: Store a high level entry (like: „rename **a** to **b**“)

Journal

Physical vs logical logging

What data could you log in the journal?

- **Logical logging**: Store a high level entry (like: „rename **a** to **b**")
- **Physical logging**: Store the file system blocks that will be modified

What might be problems with the journal presented before?

Log Structured File System

What might be problems with the journal presented before?

You write *every block twice*!

What might be problems with the journal presented before?

You write *every block twice*! How could you make that less painful?

What might be problems with the journal presented before?

You write *every block twice*! How could you make that less painful? Only journal *metadata*.

What might be problems with the journal presented before?

You write *every block twice*! How could you make that less painful? Only journal *metadata*.

How could you design a safe system that only needs to write once?

What might be problems with the journal presented before?

You write *every block twice*! How could you make that less painful? Only journal *metadata*.

How could you design a safe system that only needs to write once?

A Log Structured File System (Copy on Write). What is the core idea there?

What might be problems with the journal presented before?

You write *every block twice*! How could you make that less painful? Only journal *metadata*.

How could you design a safe system that only needs to write once?

A Log Structured File System (Copy on Write). What is the core idea there?

Only write to unused blocks.

Log Structured File System

What might be problems with the journal presented before?

You write *every block twice*! How could you make that less painful? Only journal *metadata*.

How could you design a safe system that only needs to write once?

A Log Structured File System (Copy on Write). What is the core idea there?

Only write to unused blocks.

Then you can *atomically* update indexing datastructures to point to the new blocks. If it crashes, you either have *all* of the new state or *exactly* the old state.

Letztes Tut

Das letzte Tut ist *nächste Woche*

Das wird ein Wiederholungstut von Sachen, die euch noch irgendwo unklar sind oder über die ihr nochmal reden wollt.

$\Rightarrow$ Daher bitte bis nächste Woche Fragen an mich per Mail
peter.bohner@student.kit.edu