File Systems

2024 Semester 2 COMPSCI 340: Operating Systems
Talia Xu

Lecture 1
1.0.0

To read a file

The high level idea

- Perform a path lookup
- Traverse the path until we get the file's inode
- The root inode is typically 2 in a Unix file system
- Once we have the inode of the file, we can start to read its data

Example: We want to read an 8 KB file /foo/bar

To read a file

Example: We want to read an 8 KB file /foo/bar

First we need to open the file.

Operation	data bitmap	inode Bitmap	root inode	foo's inode	bar's inode	root data	foo data	bar data[0]	bar data[1]
open(bar)									

To read a file

Example: We want to read an 8 KB file /foo/bar

Now we have the file open.

Each block is 4 KB, we must perform read twice.

Operation	data bitmap	inode Bitmap	root inode	foo's inode	bar's inode	root data	foo data	bar data[0]	bar data[1]
read()									
read()									

To write a file

The high level idea

- Similar to read – need to traverse the path to the file
- Unlike read – may need to allocate a file/block
- Block allocation updates multiple structures

Example: We want to create and write to an 8 KB file /foo/bar

To write a file

Example: We want to create and write to an 8 KB file /foo/bar

Operation	data bitmap	inode Bitmap	root inode	foo's inode	bar's inode	root data	foo data	bar data[0]	bar data[1]
create(bar)									

To write a file

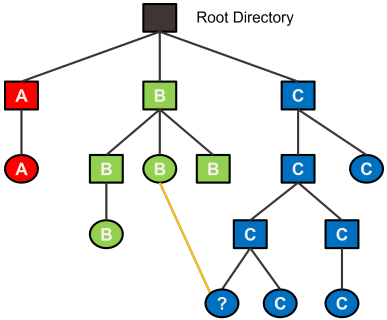
Example: We want to create and write to an 8 KB file /foo/bar

File now exists but it is empty.

We need to perform 2 writes of 4 KB each.

Operation	data bitmap	inode Bitmap	root inode	foo's inode	bar's inode	root data	foo data	bar data[0]	bar data[1]
write()									
write()									

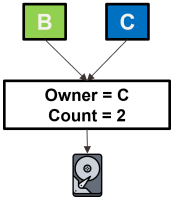
Sharing a file



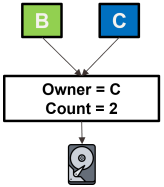
What are the different ways User B and User C can share a file?

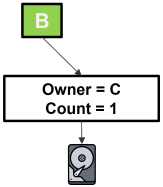
Sharing a file

Hard Links are pointers to inodes.



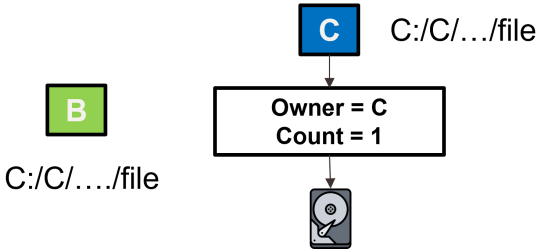
Deleting a file only removes one hard link.
The file can still be accessed





Sharing a file

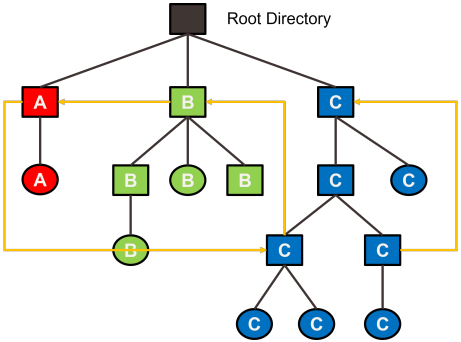
Soft links are paths to another file.



When resolving the file, the file system is redirected somewhere else, so:

- Soft link targets do not need to exist
- Soft link targets can be deleted without notice of the soft link
- Unresolvable soft links lead to an exception

Cycles in Directory Graph



Why are cyclic loops a problem?
How do we solve this?

Filesystem Example Problem

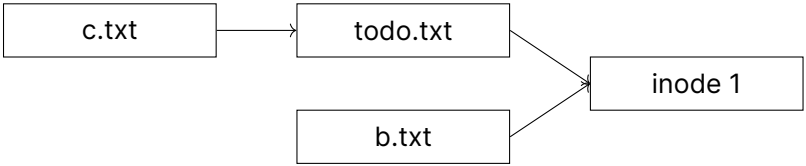
```
touch todo.txt
ln todo.txt b.txt
ln -s todo.txt c.txt
mv todo.txt d.txt
rm b.txt
```

How does the FS look like before and after the mv and rm commands?

Filesystem Example Solution (1)

```
touch todo.txt
ln todo.txt b.txt
ln -s todo.txt c.txt
mv todo.txt d.txt
rm b.txt
```

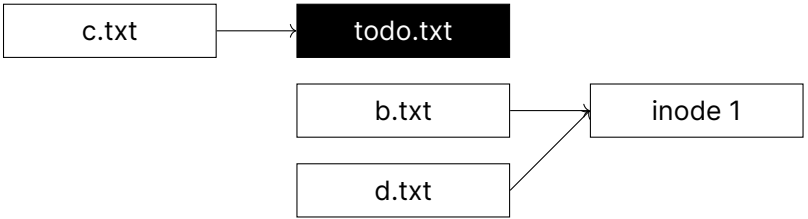
Before mv:



Filesystem Example Solution (2)

```
touch todo.txt
ln todo.txt b.txt
ln -s todo.txt c.txt
mv todo.txt d.txt
rm b.txt
```

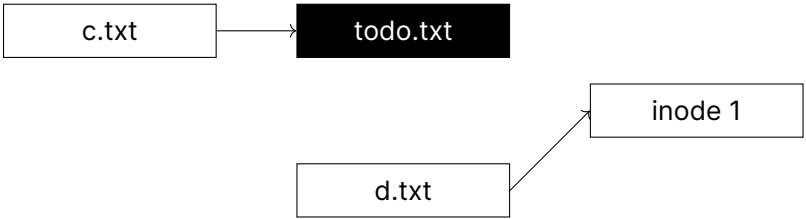
After mv:



Filesystem Example Solution (3)

```
touch todo.txt
ln todo.txt b.txt
ln -s todo.txt c.txt
mv todo.txt d.txt
rm b.txt
```

After rm:



What Data is Stored in an inode?

a Filename
b Containing Directory name
c File Size
d File type
e # of soft links to file
f location of soft links
g # of hard links to file
h location of hard links
i access rights
j timestamps
k file contents
l ordered list of data blocks

What Data is Stored in an inode? Solution

a Filename *No. Names are stored in directories*

b Containing Directory name *No. File can be in multiple dirs*

c File Size *Yes*

d File type *Yes*

e # of soft links to file *No (they are unknown)*

f location of soft links *No (they are unknown)*

g # of hard links to file *Yes (to know when to erase the file, check `stat`)*

h location of hard links *No (they are unknown to the inode)*

i access rights *Yes*

j timestamps *Yes*

k file contents *Sometimes*

l ordered list of data blocks *Yes, by definition*

Symbolic and Hard Links - More Practise

What is the output?

```
1    $ echo "hello world" > my_file
2    $ cat my_file
3    $ ln -s my_file my_symbolic_link
4    $ ln my_file my_hard_link
5    $ cat my_symbolic_link
6    $ cat my_hard_link
7    $ rm my_file
8    $ cat my_symbolic_link
9    $ cat my_hard_link
10   $ cat my_file
11   $ echo "hello world again!" > my_file
12   $ cat my_symbolic_link
13   $ cat my_hard_link
```