Exploring the Filesystem

Icon made by Freepik from www.flaticon.com

Files

Everything in Linux is a represented as a file

- - : regular file.
- d : **directory**.
- c : **character device** file.
- b : **block device** file.
- s : local socket file.
- p : named pipe.
- l : symbolic link.

How do we keep track of our files?

I want to read the file foo.txt that I wrote to disk yesterday

... where does it begin?

... where does it end?

... who is allowed to read it?

... what type of file is it?

We need a way to keep track of these things!

Inodes

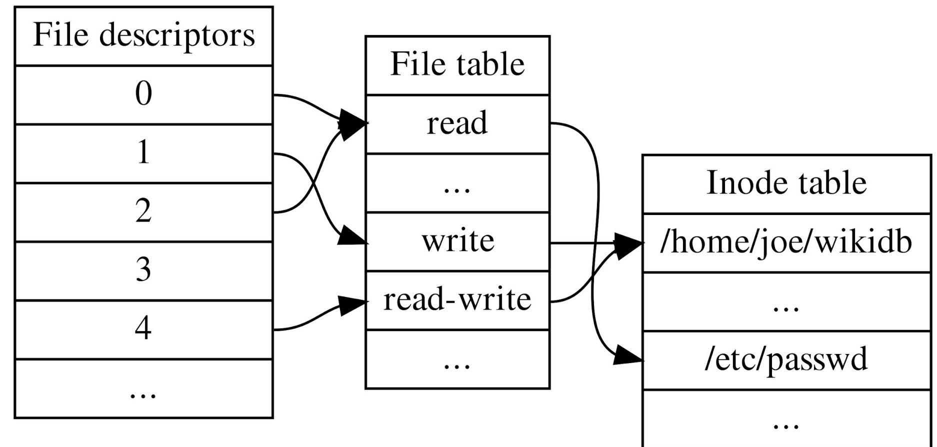
*“In truth, I don't know either….. The **I** probably refers to **Index**”*
- Dennis Ritchie

Every file has an inode which persists on disk

Directories map names to indoes

Inodes contain info about files:

- Device ID
- Inode number
- Filemode (permissions)
- Links
- UID / GID
- Device ID
- Size
- Timestamps
- I/O Blocksize
- Number of blocks



By Qwertyus - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=38970813

Bitmaps

Inodes and data blocks persist on disk

We need to know whether inodes and data blocks are free or allocated

- inode bitmap for the inode region
- data bitmap for the data region
- 0 if the region is free, 1 if allocated

The crash-consistency problem

Every time we write to a disk we need to update the inode, data block(s) and bitmaps.

What happens if we crash when we've updated on of these structures but not all?
Chaos

Solution: Journaling

- Write the contents of a transaction to a log (journal commit) before writing to the real structures (checkpoint)

Filesystems

ext4 - Journaling/log based file system. Very common as the main fs for linux

tmpfs - File system on the RAM. Common for temporary storage

Vnodes

Virtual (in memory) representation of inode

Provide file system transparency: We no longer have to consider which type of file system the machine is using

inode nr 0 - null inode

inode nr 1 - bad records

inode nr 2 - root in the file system

Soft & Hard Links

Soft (symbolic) link

Pints to another name in the filesystem.

Hard link

Another entry in the directory table.

If you delete the original file, the soft link is worthless but the hard link isn't.

```
→  test git:(master) ✗ ls -la
total 12
drwxr-xr-x 2 hrabo users 4096 11 dec 14.16 .
drwxr-xr-x 4 hrabo users 4096 11 dec 14.15 ..
-rw-r--r-- 1 hrabo users  338 11 dec 14.15 freq.dat
→  test git:(master) ✗ ln -s freq.dat s-linked
→  test git:(master) ✗ ls -la
total 12
drwxr-xr-x 2 hrabo users 4096 11 dec 14.16 .
drwxr-xr-x 4 hrabo users 4096 11 dec 14.15 ..
-rw-r--r-- 1 hrabo users  338 11 dec 14.15 freq.dat
lrwxrwxrwx 1 hrabo users    8 11 dec 14.16 s-linked -> freq.dat
→  test git:(master) ✗ ln freq.dat h-linked

→  test git:(master) ✗ ls -lai
total 16
4470984 drwxr-xr-x 2 hrabo users 4096 11 dec 14.16 .
4472298 drwxr-xr-x 4 hrabo users 4096 11 dec 14.15 ..
4470982 -rw-r--r-- 2 hrabo users  338 11 dec 14.15 freq.dat
4470982 -rw-r--r-- 2 hrabo users  338 11 dec 14.15 h-linked
4470986 lrwxrwxrwx 1 hrabo users    8 11 dec 14.16 s-linked -> freq.dat
```

Exam Questions

Exam question 1

If we want to list the content of a directory we can use the library procedure `opendir()`. Which information can we access directly form the structure pointed to by `entry` in the code below? Describe three important properties. Which information can we not find and where could this information be found?

```
int main(int argc, char *argv[]) {

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {

    // what information do we have?

  }
}
```

Exam question 1

5.1 list the content of a directory [2 points]

If we want to list the content of a directory we can use the library procedure `opendir()`. Which information can we access directly form the structure pointed to by `entry` in the code below? Describe three important properties. Which information can we not find and where could this information be found?

```
int main(int argc, char *argv[]) {

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {

    // what information do we have?

  }
}
```

Answer: We can find name, type and inode number directly in this directory entry. The rest of the information is available in the inode.

Exam question 2

5.2 remove a file [2 points]

If we us the command `rm` we will not remove a file, rather remove a hard link to a file. When is the file it self removed? How is this handled?

Exam question 2

5.2 remove a file [2 points]

If we us the command `rm` we will not remove a file, rather remove a hard link to a file. When is the file it self removed? How is this handled?

Answer: When using rm the inode is removed from the disk, while the actual data block remains. This data is only overwritten when it is needed later during the execution.

Exam question 3

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that am additional data block is needed. Which structures are updated and which changes are made?

Exam question 3

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that am additional data block is needed. Which structures are updated and which changes are made?

Answer:
We need to update the **bitmap** for used data blocks and mark the newly allocated data block as taken.
We also need to update the **Inode** for the file so that it includes the newly allocated data block.

Exam question 4

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that an additional data block is needed. When we perform the operations on disc, we only succeed in updating the inode but not the bit maps nor the selected data block before we crash.

If we do not detect the error when we restart, which problems will we have and what could happen?

Exam question 4

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that an additional data block is needed. When we perform the operations on disc, we only succeed in updating the inode but not the bit maps nor the selected data block before we crash.

If we do not detect the error when we restart, which problems will we have and what could happen?

Answer: We will have a data block that is allocated to an inode but the data block contains garbage and it is marked as free in the bit-maps. If we read from the file we will read garbage but worse if we use the data block for another file. This new file will then write its data to the block that can then be over written when we write to the first file. If the data block is used to represent a directory, this could of course result in total chaos.

Exam question 5

You are browsing you filesystem using the code displayed on the right.

Suddenly you get a segmentation fault.

What went wrong?

```
#include <stdio.h>
#include <dirent.h>

int main(int argc, char *argv[]) {

  if( argc < 2 ) {
    perror("usage: myls <dir>\n");
    return -1;
  }

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {
    printf("\tinode: %8lu", entry->d_ino);
    printf("\tname: %s\n", entry->d_name);
  }

  return 0;
}
```

Exam question 5

You are browsing you filesystem using the code displayed on the right.

Suddenly you get a segmentation fault.

What went wrong?

Answer: The opendir function returns a null pointer if the user does not have permission to read the file.

```
#include <stdio.h>
#include <dirent.h>

int main(int argc, char *argv[]) {

  if( argc < 2 ) {
    perror("usage: myls <dir>\n");
    return -1;
  }

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {
    printf("\tinode: %8lu", entry->d_ino);
    printf("\tname: %s\n", entry->d_name);
  }

  return 0;
}
```

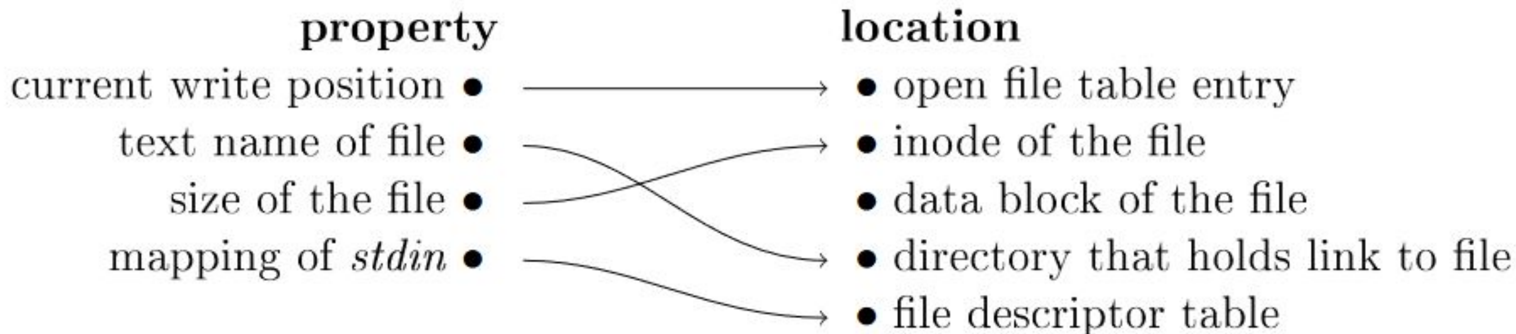
Exam question 6

A file has many properties; where do we find the below listen properties? Connect the properties to the left with the correct locations to the right. Several properties might be found at the same location but each property is only found at one location (many to one).

property	**location**
current write position •	• open file table entry
text name of file •	• inode of the file
size of the file •	• data block of the file
mapping of *stdin* •	• directory that holds link to file
	• file descriptor table

Exam question 6

A file has many properties; where do we find the below listen properties? Connect the properties to the left with the correct locations to the right. Several properties might be found at the same location but each property is only found at one location (many to one).



Exam question 7

A directory can hold links of different types, describe the type of the following five links:

```
drwxrwxr-x 2 johanmon johanmon 4096 dec 21 17:43 bar.doc
-rwxrwxr-x 1 johanmon johanmon 8464 dec 21 22:25 cave
lrwxrwxrwx 1 johanmon johanmon    7 dec 21 17:43 foo.pdf -> ./gurka
-rw-rw-r-- 1 johanmon johanmon    7 dec 21 17:42 gurka
prw-r--r-- 1 johanmon johanmon    0 dec 21 22:25 sesame
```

Exam question 7

A directory can hold links of different types, describe the type of the following five links:

```
drwxrwxr-x 2 johanmon johanmon 4096 dec 21 17:43 bar.doc
-rwxrwxr-x 1 johanmon johanmon 8464 dec 21 22:25 cave
lrwxrwxrwx 1 johanmon johanmon    7 dec 21 17:43 foo.pdf -> ./gurka
-rw-rw-r-- 1 johanmon johanmon    7 dec 21 17:42 gurka
prw-r--r-- 1 johanmon johanmon    0 dec 21 22:25 sesame
```

Answer: *bar.doc* is a directory, *cave* is an executable file, *foo.pdf* is a symbolic (soft) link, *gurka* is a regular file and *sesame* is a *pipe*

Exam question 8

Assume that we have a journal based file system. Which of the following statements below is/are correct and which ones is/are false - explain why.

- It is important the the transactions are checkpointed in the same order as theyr were created.
- A transaction is considered to be valid ones it is written to the journal.
- In a restart after a crash, we must be careful not to checkpoint a transaction twice.

Exam question 8

Answer:

- Correct:if several transactions changes the same block the result is the last performed.
- Correct: a halfway written transaction is not valid, a fully written transaction is. This is independent of if we have performed the checkpoint, changing the content of the real blocks.
- False: since the operations are idempotent it does not matter if we perform a transaction twice (the only thing that is important is that they are performed in the right order).