# 1 File systems

1.1. Name two factors that are important in the design of a file system.

Index structure and size of file blocks are important factors when designing a file system.

1.2. Name some examples of file metadata.

There are for instance File owner, access permissions, when the files were created, when they were last changed, etc..

# 2 Files and directories

2.1. Consider a Fast File System (FFS) like Linux's ext4.

(a) Explain the difference between a hard link and a soft link in this file system. What is the length

of the content of a soft link file?

A hard link is the mapping from the folder directly to the underlying folder/file, while a soft link is a mapping between two different files.

(b) What is the minimum number of references (hard links) for any given folder?

A folder has a minimum of 2 links on creation, one link to itself and one link to its parent folder.

(c) Consider a folder /tmp/myfolder containing 5 subfolders. How many references (hard links) does

it have? Try it yourself on a Linux system and include the output. Use ls -ld /tmp/myfolder

to view the reference count (hint, it's the second column in the output).

It should have 7 hard links, since all folders have 2+(amount of sub-folders) links.

(d) Explain how spatial locality is achieved in a FFS.

By fragmenting small files and storing large files continuously.

2.2. NTFS - Flexible tree with extents

(a) Explain the differences and use of resident versus non-resident attributes in NTFS.

When the file’s attributes can fit within the MFT file record it is a resident attribute, while a non-resident has file attributes that are too large to fit within the MFT so they have to be stored separately and pointed to by the MFT resident data.

(b) Discuss the benefits of NTFS-style extents in relation to blocks used by FAT or FFS.

Flexible due to MFT blocks, block pointers that covers several blocks, more reliable due to journaling.

2.3. Explain how copy-on-write (COW) helps guard against data corruption.

Copy-on-write creates a tempral link to the same file instead of creating a new file when making a copy, so the system doesn’t need to generate a new file until one of them is changed.

# 3 Security

3.1 Authentication

(a) Why is it important to hash passwords with a unique salt, even if the salt can be publicly known?

This makes it harder to brute-force a password using pre-generated password-tables and increases security.

(b) Explain how a user can use a program to update the password database, while at the same time

does not have read or write permissions to the password database file itself. What are the caveats

of this?

This is possible because the encryption on the password database is local on each users system, so they can always access their info even if the server is updated to a new system.

3.2. Software vulnerabilities

(a) Describe the problem with the well-known gets() library call. Name another library call that is

safe to use that accomplishes the same thing.

Gets() offers no protection against buffer overflow, a better function is fgets() which is a bit harder to use but doesn’t cause the same problems as gets()

(b) Explain why a microkernel is statistically more secure than a monolithic kernel.

In a microkernel the user and kernel services are kept in separate address spaces, while in a monolithic kernel everything is kept in the same address space. This means that its easier for a user program to get access to things it shouldn’t in a monolithic kernel.