Usman Ghafoorzai & Jonas Johansen

NTNU

OBL4-OS

1 File Systems

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

Two factors that are important in the design of a file system are **storage efficiency** and **readability**. That is, ensuring efficient space utilization for both small and large files – including using small blocks for small files and methods like contiguous allocation for large files to improve access-speed. With reliability, it is meant safeguarding data integrity and ensuring recovery from crashes or errors. Some techniques such as journaling or data replication can help maintain reliability and prevent data loss.

1.2 Name some examples of file metadata.

Some examples of a file's metadata include the file's **size**, its **modification time**, its **owner** and its **security information** such as whether it may be read, written, or executed by the owner or by other users.

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?

Soft and hard links are different ways of referring to data in a file system. Soft links, also known as symbolic links, acts as a shortcut or reference to files and directories. Hard links on the other hand are direct references to the inode. This means that if we have a file named a.txt and have both a soft and hard link of it. Deleting the a.txt renders the soft link unusable as it points to nothing, while the hard link still works because it points to the memory of that file.

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

The minimum number of hard links that exist for any folder in a unix system is 2. This is the directory and the self-reference. The directory itself has a hard link pointing to itself, representing the directory's name within its parent directory. For example, a directory named mydir has a link to itself from the parent directory. The . link (Self-reference) contains a . entry, which is a hard link pointing back to itself. This . entry allows processes to reference the current directory within its own context.

(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).

We can look at the answer we gave in the previous subtask saying that at minimum there are 2 hard links in any given folder. With this we can count that if we have one root folder and five subfolders that we must have 6*2 amount of hard links present at least. From testing we can see that this does in fact coincide with the theory.

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

To keep track of the data blocks that belong to each file , FFS uses a fixed, asymmetric tree called a multi-level index which provides good on-disk layout.

2.2 NTFS – Flexible tree with extents

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

Whereas FFS tracks file blocks with a fixed tree, NTFS and many other recent file systems such as Linux ext4 and btrfs track extents with flexible trees. The roots of these trees are stored in a master file table that is like the FFS' inode array. NTFS' master file table (MFT) stores an array of 1KB MFT records, each of which stores a sequence of variable-size attribute records. NTFS uses attribute records to store both data and metadata. Both are just considered attributes of a file. Some of these

attributes may be too large to store in an MFT record. An attribute can therefore be either a resident or non-resident. A resident attribute stores its contents directly in the MFT record while a non-resident attribute stores extent pointers in its MFT records and stores its content in those extents.

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

As we mentioned above data and metadata stored in a NTFS file system is different compared to the block architecture used in FFS and FAT. NTFS uses a extent-based system that allows it to handle a wide range of file sizes efficiently. An extent is a contiguous sequence of disk blocks that represents a portion of a file, which enables NTFS to store data more flexibly and with less fragmentation compared to systems like FAT and FFS, which track each file as a series of individual blocks.

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

Copy-on-write is a technique used in file systems to help guard against data corruption. The way it works is that when data or metadata needs to be modified, COW does not overwrite the existing copy. Instead, it creates a separate copy which is intended for modification. The system then writes these changes to the new copy. Once all the changes have been added to the copy, an atomic operation occurs which takes the pointer to the original data and changes it to point to the new copy. Since it is an atomic operation, either everything happens, or nothing happens. So, if a step fails, then it doesn't update the pointer, preventing partial updates. After the new data has been confirmed and committed, the original data can be marked as free space, allowing it to be reused.

3 Security

3.1 Autentication

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

The reason it is important to salt the hashed passwords regardless of the salt being publicly known is that when we hash passwords, non-unique passwords gain the same value, making it easier for attackers to deduce what the password may be. With salting this does not become an issue as same passwords result in a different hash.

(b) Explain how a user can use a program to update the password database file itself. What are the caveats of this?

A program can be used to update the password database file by writing a new entry or modifying an existing one. This, of course, requires the program to have sufficient privileges (for example root access) since the password database is typically protected. The program might edit the file directly or use system utilities like **passwd**. Some caveats of this is:

Security Risks: direct editing might expose sensitive data if not handled securely. **File Integrity**: Errors during editing (for example crashes) can corrupt the database. **Access Control**: Misuse of elevated privileges can introduce vulnerabilities.

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.

The **gets()** library call is unsafe since it does not check the length of the input buffer, allowing for buffer overflow vulnerabilities. Consequently, this can lead to **memory corruption** – overwriting adjacent memory locations. **Security Risks** – exploits such as injecting malicious code. A safer alternative is **fgets()**, which

allows you to specify the maximum buffer size and thus prevents buffer overflow by limiting the number of characters read.

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

A microkernel is statistically more secure than a monolithic kernel for several reasons. First, microkernels implement only essential functionalities, such as inter-process communication, basic scheduling, and memory management, within the kernel, while most services like file systems and drivers run in user space. This results in a smaller codebase, inherently reducing the number of bugs and vulnerabilities, and minimizing the attack surface. Additionally, microkernels provide strong fault isolation by running most services in user space. If a service crashes, it does not compromise the kernel or other services. In contrast, monolithic kernels integrate all services within the kernel itself, meaning a failure in any module, such as a buggy device driver, can crash the entire system.

The modular design of microkernels also enhances security, as services communicate through well-defined APIs that can be monitored and restricted. In monolithic kernels, tightly integrated components make it harder to isolate or secure interactions between them. Furthermore, microkernels enforce limited privileges for user-space services, reducing the potential damage from a compromised component. On the other hand, monolithic kernels run most of their code with full system privileges, increasing the risk posed by compromised modules. Lastly, the smaller and more modular design of microkernels simplifies auditing and maintenance of security-critical parts. In contrast, the large and complex codebases of monolithic kernels are more difficult to audit comprehensively. Overall, the modularity, fault isolation, and reduced codebase of microkernels significantly enhance their security compared to monolithic kernels.