OBL4-OS

1. File systems

1.

1. Efficiency and Performance: This factor involves designing the file system in a way that optimizes access speed and data throughput. Efficiency is crucial in how the file system manages disk space, how it retrieves and stores files, and how it handles caching and buffering.

2. Reliability and integrity: A file system must be reliable, ensuring that data is not corrupted and is consistently available. This includes the implementation of features for data recovery in case of system crashes or power failures, such as journaling or versioning. Data integrity involves protecting the data from errors due to hardware malfunctions, power outages, or system crashes.

These factors are fundamental as they directly impact the usability, stability, and security of the file system and the data it manages.

2.

File metadata includes various data attributes that describe and provide information about a file, apart from the files actual content. Some examples of file metadata are: file name, file type or extension, file size, date and time stamps, permissions and file location.

2. Files and directories

1. (a) Hard Link vs Soft Link

A hard link is essentially an additional name for an existing file on the file system. It points directly to the inode (data structure) of the file, not the file name. Hard links behave like the original file. They have the same permissions, timestamps, and other attributes. Deleting the original file dose not affect the hard link, as the file’s data remains on the disk until all hard links to the inode are deleted.

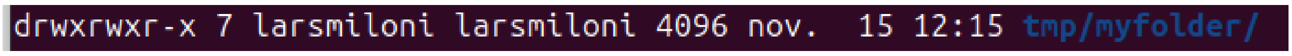
A soft link (or symbolic link) is a special file that points to another file or directory. It’s a path to another file, not a direct link to the data itself. Soft links can span file systems and can link to directories. If the original file is moved, renamed, or deleted, the soft link breaks and becomes unusable, showing a dangling link. The length of the content of a soft link file is the length of the path to the original file it points to, not the size of the original file itselfe.

(b) Minimum Number of References for a Folder

In Unix-like file systems, every folder has a minimum of two hard links. The first linkk is the name of the folder itself. The second is the . (dot) entry that refers to the folder itself from within its own directory.

(c) Reference Count of a Folder with Subfolders

In addition to the two default hard links, every subfolder within a folder adds one additional hard link to the parent folder for its (dot-dot) entry, which refers back to the parent. Thus, a folder '/tmp/myfolder' with 5 subfolders will have '2 (default) + 5 (subfolders) = 7' hard links.



```
drwxrwxr-x 7 larsmiloni larsmiloni 4096 nov.  15 12:15 tmp/myfolder/
```

(d) Spatial Locality in FFS

Spatial locality in file systems like FFS is achieved through the way data is organized and stored. FFS uses techniques such as blocking (storing files in blocks of fixed size) and clustering (grouping several blocks together). These methods ensure that related data is stored close together on the disk. Additionally, FFS often allocates inodes and data blocks for files in the same or nearby cylinder groups, reducing the head movement of the disk, thus increasing access speed. This organization enhances the performance, especially for operations that access multiple contiguous blocks of data, like reading a large file.

This should provide a comprehensive overview of the concepts and operations related to the Fast File System and its mechanisms like hard and soft links, as well as spatial locality.

2. (a) Resident vs Non-Resident Attributes in NTFS

Resident attributes: Stored directly in the files Master File Table (MFT) record. Used for small data like file names, timestamps. Quick to access but limited by MFT record size.

Non-Resident attributes: Used for larger file data that dosen't fit in the MFT record. Data is stored elsewhere on the disk, with pointers in the MFT. Suitable for the file's content.

(b) Benefits of NTFS-Style Extents over FAT or FFS Blocks

Reduced Fragmentation: NTFS extents store large files in contigous clusters, minimizing fragmentation.

Efficient Large File Handling: Better at managing large files, avoiding numerous block pointers.

Performance Improvement: Faster access to large files due to contiguous data storage.

3. Copy-on-write (COW) guards against data corruption by ensuring that changes are made to a copy of the data, rather than the original. If an error occurs during the modification, the original data remains unaltered, thus maintaining data integrity and preventing partial or corrupt writes. This technique is especially valuable in preventing file system corruption during unexpected system failures.

3. Security

1. (a) Hashing Passwords with a Unique Salt

Prevents Rainbow Table Attacks: A unique salt makes each password hash unique, hindering attackers ability to use precomputed tables for cracking hashes.

Avoids Duplicate Hashes: Different salts for identical passwords prevent the same hash from appearing, increasing security.

(b) Updating Password Database Without Direct Access

Dedicated Program: A program with appropriate permissions allows users to update their passwords. Users interact with this program, not the database directly.

Caveats: The program must be secure against vulnerabilities and strictly validate user input. Proper logging and auditing are essential to monitor for unauthorized access or modifications.

2. (a) Problem with ‘gets()’ Library Call

Buffer Overflow: The primary issue with the ‘gets()’ function in C is that it does not check the size of the buffer into which it reads input. This can lead to buffer overflow, where data exceeds the buffer’s capacity, potentially overwriting adjacent memory. This vulnerability can be exploited to execute arbitrary code or crash the program.

Safe Alternative: A safer alternative to ‘gets()’ is ‘fgets()’. ‘fgets()’ takes an additional parameter specifying the buffer size, thereby preventing buffer overflow by limiting the amount of input read.

(b) Security of Microkernel vs. Monolithic Kernel

Smaller Attack Surface: Microkernels have a smaller code base and functionality in the kernel space. Reducing the attack surface compared to monolithic. In a microkernel, most services run in user space, isolated from the kernel, making it harder to exploit kernel vulberabilities.

Better Isolation of Components: Microkernels isolate system components better. If one component fails or is compromised, it’s less likely to impact the entire system.

Easier to Audit and Secure: Due to their smaller and more modular design, microkernels are easier to audit for security vulnerabilities. Monolithic kernels, with their larger and more complex codebase, are more challenging to thoroughly secure.

Fault Tolerance: Microkernels offer better fault tolerance. Errors in one component often don’t crash the entire system, enhancing overall system security and stability.