Assignment 4

1 File systems

1. **Name two factors that are important in the design of a file system.**
    a. **Reliability**: The data which is stored in a file system should be safely stored even if the system crashes or the power is turned off. In many cases the data should survive even it the device is damaged.
    b. **Performance**: Programs which uses huge amounts of data, have to be able to access data fast. If the files cannot be delivered fast, then the users will experience annoying delays which lowers the user-satisfaction.
2. **Name some examples of file metadata.**
    File size, modification time, owner, access-control information

2 Files and directories

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?**
    Hard links are multiple file directory entries that map different path names to the same file-number. Soft links on the other hand are directory entries that maps to another directory entries. The size of a soft link is the number of characters in the path it points to.
    b. **What is the minimum number of references (hard links) for any given folder?**
    It's usually at least two hard links in any folder: "." and "..". "." is a relative reference for the current directory and ".." is a relative reference for the parent directory. This can be viewed by typing "mkdir <dir-name>" and then immediately "ls -lhaF" which lists all references.
    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 has 7 references
    d. **Explain how spatial locality is acheived in a FFS.**
    It uses a *block group placement* and *reserve space* to get good spatial locality for a wide range of workloads.
        i. *Block group placement*: FFS places data to optimize for the common case where a file's data blocks, a file's data and metadata, and different files from the same directory are accessed together.
        ii. *Reserved space*: To handle cases where a disk is nearly full and there is little opportunity for the file system to optimize locality, FFS reserves a fraction of the disk's space (for ex. 10%) which it hides from applications. Only administrators are allowed to allocate news blocks when everything else is in use, which is useful for cleaning things up. The small disk capacity sacrifice gives reduced seek times in return.

2. **NTFS - Flexible tree with extents**
    a. **Explain the differences and use of resident versus non-resident attributes in NTFS.**
    A resident attribute stores its contents directly in the MFT (Master File Table) record while a non-resident attribute stores extent pointers in its MFT record and stores its contents in those extents. Resident attributes are used if the content is small enough to be store right inside the MFT record, else non-resident attributes are used.
    b. **Discuss the benefits of NTFS-style extents in relation to blocks used by FAT or FFS.**
    Rather than tracking individual file blocks, NTFS tracks extents, variable-sized regions of files that are each stored in a contiguous region on the storage device. The blocks used by FAT and FFS on the other hand may be scattered, especially when the disk fills. Blocks are also inefficient for tiny files as a 1 byte file requires both an inode and a data block. NTFS is more flexible as it can store small files directly in the MFT record, and larger files can use extents.
3. **Explain how copy-on-write (COW) helps guard against data corruption.**
    When updating an existing file, copy-on-write (COW) file systems do not overwrite the existing data or metadata; instead, they write new versions to new locations. Not updating the original location eliminates the risk of a partial update or data corruption during a power failure. It also provides features like backup and restoration of data.

3 Security

1. **Authentication**
    a. **Why is it important to hash passwords with a unique salt, even if the salt can be publicly known?**
    Salts defend against a pre-computed hash attack, for example rainbow tables. Since salts are unique in each case, they also protect commonly used passwords by making all salted hash instances for the same password different from each other.
    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?**
    In cases where a user wants to update his password, the program could can use multiple methods to first verify that the user should be allowed to do this. It could for example email a link to the user which contains a token that again tells the program that this is the correct user. The program could also require the user to enter the old password first, although that's difficult if the user has forgotten it. When the program knows that it's the correct user who's trying to update a password, it can act like a middleman which receives the password from the verified user and then updates the database.
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()* function is a common source of buffer overflow vulnerabilities and should never be used. Programs running with elevated privileges, including programs that are outward facing, can be used for privilege escalation or to launch a remote shell. There are two alternative functions that can be used: *fgets()* and *gets_s()*.
    b. **Explain why a microkernel is statistically more secure than a monolithic kernel.**

Usually the capabilities of microkernels are more restricted then those of a "general purpose" monolithic kernels and thus contain less bugs purely because their codebase is smaller. Also, the separation of drivers from the kernel means that a driver can’t crash or modify the kernel. Failing drivers can also be restarted without the rest of the system being affected, a failing microphone driver in an aircraft will for example not crash the entire aircraft system. It’s also easier to update a security flaw in a microkernel than having to update a monolithic kernel.