Vika 14

Sturla Freyr Magnússon

Explain the function of file systems.

File systems are responsible for organizing, managing and storing data on storage devices. They also handle metadata, access control and ensure data reliability and consistency while providing a structured way for the OS, applications and users to access and manipluate files and directories

Describe the interfaces to file systems.

- POSIX operating systems:
  - Open an existing file:
    `int open(const char *pathname, int flags)`
  - Create new file/Overwrite an existing file and open it in write mode:
    `int creat(const char *pathname, mode_t mode)`
    - Return value: File descriptor, -1=error
    - pathname: Name of file to open
    - flags: e.g.: `O_RDONLY`/`O_WRONLY`/`O_RDWR`=for read only/write only/read and write access, `O_APPEND`=for appending at the file end.
    - mode: Access rights for the file to be created
  - Close opened file:
    `int close(int fd)`
    - Return value: -1=error
    - fd: File descriptor of file to be closed

- Read data starting from current file pointer position:
  `ssize_t read(int fd, void *buf, size_t count)`
- Write data starting from current file pointer position:
  `ssize_t write(int fd, const void *buf, size_t count)`
  - Return value: Number of actually transferred bytes, -1=error
  - fd: File descriptor
  - buf: Address in main memory where bytes should read into/write from
  - count: Number of bytes to be transferred
- Move file pointer: `off_t lseek(int fd, off_t offset, int whence)`
  - Return value: New (absolute) file pointer position, -1=error
  - fd: File descriptor
  - offset: New position for file pointer (relative or absolute according to whence)
  - whence: `SEEK_SET`=absolute, `SEEK_CUR`=relative with respect to current file pointer, `SEEK_END`=relative with respect to end of file (i.e.using negative offset)
- Further operations: `truncate`, `rename`, `unlink` (delete file or directory).

Understand memory-mapped files.

A memory-mapped file is a file, or partial file, that has had it's locay into a processes virtual address space. This means it's content can be accesses direordinar memory access (e.g. pointers) instead of much slower system calls. Memory-mapped files allow the OS to load only those pages from a file that are actually accessed.

Discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures

Access methods

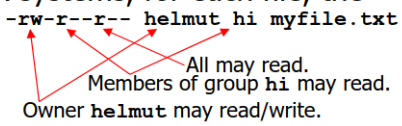
- Sequential access. Files are accessed in linear order.
- Direct access. Files are accessed using a block address or offset, enabling efficient random access.
- Indexed access. An indexed structure allows for locating data within a file which can provide faster access for some search operations.

File sharing

An OS must support file sharing e.g. system files to avoid duplication and application files to allow collaboration. It also has to protect files from being accessed by unauthorised users.

To manage sharing/protection on POSIX systems, for each file, the access rights

- read/write/execute can be granted to
- owner/user (user that created the file),
- group (each file has also a group associated: by default, the group to which the owner belongs. Groups are defined by the system administrator),
- all (any user on that system).



File locking

See file sharing

Directory structures

Directories are just files!

Directories contain files and sub-directories organized in a tree from the root directory.

- Files in different directories are independent of each other
- A path navigates to files
- "/" is the directory level seperator
- A path beginning with "/" is an absolute path otherwise it's relative.
- Hard links: Points to the block of the original inal file making the

wo indistinguishable

- Symbolic link: A special file that contains a path to the original file

Explore briefly file-system protection.

Mostly consists of the following:

- Authentication: Authenticate the identity if users.
- File and directory ownership: Each file/directory is associated with an owner, usually the user that created it.
- File locking and synchronization: Used to maintain data consistency and prevent conflicts in systems with multiple runnings processes or users.
- Encryption: Optional but useful mechanism for protecting the confidentiality of data stored in the file system.

.

Describe the details of implementing local file systems and directory structures.

File system layers

- Logical file system: manages file pointer, metadata (e.g. owner, permissions, directory contents), symbolic links.
- File-organisation module: manages in which blocks of the device the file contents is stored (allocation methods) and where free blocks can be found on the device (free space management)..- Basic block I/O: Handles buffers for blocks that are waiting to be transferred to/from storage device, caches blocks.
- I/O Control: Device driver that is specific to underlying device controller.
- Device: actual I/O device and controller.

Discuss block allocation methods and free-space managemen

- Contigious allocation. Is problematic because if a file needs to grow subsequent block may be allocated to other files.
- Extents: If initial chunk of contiguous blocks is full, use additional contiguous chunks (an extent) that start at a new location and can be added to the already existing extents of a file.
- Linked and indexed allocation: For each individual fixed-sized block of a file, it is always (even if these blocks are contiguous) stored in the metadata where it is located in the file system. FAT(linked) and I-node(indexed) are of these types.

Explore file system efficiency and performance issues.

- Searching FAT for free clusters: Search the whole FAT for special number indicating a free cluster. Complexity O(n) n=number of clusters.
- Searching I-node for free block:
  - I-node only keeps track of allocated blocks, not free blocks.
  - Use additional bitmap where one bit indicates wether corresponding block is free or allocated
  - Complexity O(n) n=number of bits in bitmap
  - Size of bitmap: as many bits as file system has blocks
- Defragmentation/Compaction: Move blocks/clusters by copying them to create contigious free space for future files.

Look at recovery from file system failures

- Consistency checking: Regularly check file system consistency to prevent that a possible corruption affects even further data.
  - Often done at the next system boot after a system crashed
  - Tools for checking file system consistency: fsck on Unix, chkdsk on MS Windows.
- Journaling/Log-based Transaction-oriented File Systems
  - Journaling file systems can avoid inconsistencies to occur at all.
  - At least, inconsistencies due to power outage/system crash while writing data can be prevented
  - Journaling file systems are state of the art in all modern file systems e.g.(Microsoft NTFS, all modern POSIX/Unix-like file systems, but not FAT).
  - Based on the concept of transactions: Either write all data or no data!
  - I.e. either old file system state or new file system state.
  - Three step approach: announce action, commit action, acknowledge action.
- Journaling/Log-Based approach:
  - Write information about intended changes to an intermediate buffer

- If journal full: Write changes logged in journal to actual storage locations
- Delete entry from journal.