K22 – Operating Systems: Design Principles and Internals

Fall 2025 @dit

Vaggelis Atlidakis

Lecture 19

References: Similar OS courses @Columbia, @Stanford, @UC San Diego, @Brown, @di (previous years); and textbooks: Operating Systems: Three Easy Pieces, Operating Systems: Principles and Practice, Operating System Concepts, Linux Kernel Development, Understanding the Linux Kernel

Overview

- We'll start from hardware and follow a question-oriented approach
  - ~~Intro [Q: What is an OS?]~~
  - ~~Events [Q: When does the OS run?]~~
  - ~~Runtime [Q: How does a program look like in memory?]~~
  - ~~Processes [Q: What is a process?]~~
  - ~~IPC [Q: How do processes communicate?]~~
  - ~~Threads [Q: What is a thread?]~~
  - ~~Synchronization [Q: What goes wrong w/o synchronization?]~~
  - ~~Time Management [Q: What is scheduling?]~~
  - ~~Memory Management [Q: What is virtual memory?]~~
  - **Files [Q: What is a file descriptor?]**
  - Storage Management [Q: How do we allocate disk space to files?]

* Basic (H/W & S/W)
* **Abstractions**
* Primitives
* **Mechanisms**

Overview

- Files
  - **Q1**: What is a file?
  - **Q2**: Why need a file?
  - **Q2**: Necessary file metadata?
  - **Q3**: What is a directory?
  - **Q4**: What is a POSIX file descriptor?
  - **Q5**: File-related POSIX operations?
  - **Q5**: Safely updating the contents file?
  - **Q6**: File accesses patterns?

What is a file?

"An object that can be written to, or read from, or both, with data and attributes, such as access perms and type." (POSIX def. 3/139.)

> A named byte-array persistent across reboots

- **Regular files:** Contain user data in text or binary format
- **Special files:** Devices for char-by-char (e.g., /dev/tty) or block-based (e.g., /dev/sda) data transfers
- **Named Pipes:** First-in-first-out IPC mechaninsm
- **Sockets:** Endpoint for network communication or IPC
- **Directories:** Contains a list of file names
- **Symbolic links:** A pointer or shortcut to another file or directory

Why need a file?

"An object that can be written to, or read from, or both, with data and attributes, such as access perms and type." (POSIX definition 3/139.)

> A named byte-array **persistent** **across reboots**

- Helps identify data by using natural language names
- Abstracts the details of the underlying storage devices
- First and only persistent abstraction
  - >> Persists across reboots
  - >> Persists across power failures
  - >> Storage devices healthy and filled w/ electricity? Life is good

Necessary file metadata?

```
➜  git:(master) ✗ /bin/ls -hlia ept.patch
32078924 -rw-r--r--  1 parallels parallels 18K Dec 30 23:28 ept.patch
```

- File Identifier: Identifies file within file system (inode in Linux)
- Access Control List (ACL): Controls users and allowed accesses
- Owner and Group: Used along with ACLs for permission checking
- Size: File size in bytes, KiB, and so on
- Timestamp: Time of last modification
- Filename: The name of the file, in human-readable format

What is a directory?

"A file that contains directory entries; that is, objects that associate filenames with a files" (POSIX definition 3/103.)

> Conceptually: A hierarchical organization technique, based on an acyclic-graph hierarchy: e.g., A/B, implies that file or directory B, lives under its parent directory A.

> Technically: each directory is a file whose data is a list of <*filename*, *index*> pairs.

> Root "/" directory: Special directory, root of the hierarchy

File-related POSIX syscalls

int open (const char *pathname, int flags, ...)

› Given a file pathname, open() returns a non-negative process-unique inheritable open file handle integer (called a **file descriptor**), for use in subsequent syscalls.

- pathname: The name identifying the target file
- flags: Must include one of the following access modes O_RD/WRONLY or O_RDWR.
- On success, open(...) returns a non-negative integer; or, -1 is returned, if an error occurred

int rename (const char *oldpath, const char *newpath)

› Renames a file, potentially moving it between directories if required. Any other hard links to the file as well as "oldpath"-related open fds are unaffected.

- oldpath: Origin path
- newpath: Destination path
- On success, rename(...) returns zero; or, -1 is returned, if an error occurred

What is a POSIX file descriptor?

"A per-process unique, non-negative integer used to identify an open file for the purpose of file access. The values 0, 1, and 2 are referred to as standard input, standard output, and standard error." (POSIX def 3/141.)

Expensive to resolve name to identifier on each access

› **Elegant POSIX solution:** Open file before access

Brief implementation details (more later..)

1. Search directories for file name, locate and check permission
2. Read file metadata into a system-wide in-memory open files table
3. In-process integer, called file descriptor (fd), indexes the open files table
4. Processes reuses fd by passing it to the OS for subsequent file access
5. Process needs to access a new file? Will add a new integer to its fd table

File-related POSIX syscalls

int open (const char *pathname, int flags, ...)

› Given a file pathname, open() returns a non-negative process-unique inheritable open file handle integer (called a **file descriptor**), for use in subsequent syscalls.

- pathname: The name identifying the target file
- flags: Must include one of the following access modes O_RD/WRONLY or O_RDWR.
- On success, open(...) returns a non-negative integer; or, -1 is returned, if an error occurred

int rename (const char *oldpath, const char *newpath)

› Renames a file, potentially moving it between directories if required. Any other hard links to the file as well as "oldpath"-related open fds are unaffected.

- oldpath: Origin path
- newpath: Destination path
- On success, rename(...) returns zero; or, -1 is returned, if an error occurred

File-related POSIX syscalls

int unlink (const char *pathname)

> Deletes a name from the filesystem and possibly the file it refers to. If "pathname" is the last link to a file and no process has the file open, the file is deleted and the space it was using is made available for reuse.

- pathname: The name identifying the target file
- On success, unlink(...) returns zero; or, -1 is returned, if an error occurred

int truncate (const char *path, off_t length)

> Cause the regular file named by path to be resized to precisely length bytes, such that if the file previously was larger, the extra data; or, it it was previously shorter, it is extended, and the extended part reads as null bytes

- pathname: The name identifying the target file
- length: The target, new length
- On success, truncate(...) returns zero; or, -1 is returned, if an error occurred

File-related POSIX syscalls

int read (int fd, int *buf, size_t count)

› Attempts to read up to *count* bytes from the file descriptor *fd* into *buf*.

- On success, read(...) returns the number of bytes read and the file position is advanced accordingly. Zero indicates end of file, while, it is not an error, if this number is smaller than count. On error, -1 is returned, and errno is set appropriately.

int write (int fd, int *buf, size_t count)

› Attempts to write up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

- On success, write(...) returns the number of bytes written, while, zero indicates that nothing was written. On error, -1 is returned, and errno is set appropriately.

File-related POSIX syscalls

off_t lseek (int fd, off_t offset, int whence)

> Given an open *fd*, lseek repositions the respective's file *offset* according to the directive *whence* to be at position (i) "offset" (SEEK_SET); (ii) "current position" + "offset" (SEEK_CUR); or, (iii) "size of file" + "offset" (SEEK_END).

- On success, lseek() returns the resulting offset location as measured in bytes from the beginning of the file; or, -1 is returned, if an error occurred

int fsync (int fd)

> Transfers, i.e., "flushes", all modified in-kernel data and metadata of the file associated with fd to the underlying storage devive. fsync(...) blocks until the device reports that the transfer has completed.

- On success, fsync(...) returns zero; or, -1 is returned, if an error occurred

Crash-tolerant file updates

> Typical goal when dealing with files: How to safely update a file, even given the potential for a crashes or power failure to occur?

Crash-tolerant file update pattern

1. write: data -> temp_file
2. fsync: temp_file
3. rename: temp_file -> target_file [rename is an atomic operation]
4. fsync: parent_dir
5. Assert temp_file does not exist

File accesses patterns

› Sequential Access
- Data read from or written to storage in order
- Good temporal locality => Can be efficiently proactive with prefetching
- Examples: User copying files, Compiler reading / writing files

› Random Access
- Randomly accessing any block
- Poor spatial Locality => Difficult to make fast / What to prefetch?
- Used to be a bigger problem in the past (seek time and rotational delay)
- Still problematic because it undoes prefetching benefitsproactivity
- Examples: Updating records in a database file