Student ID: 230018374 **Date:** 25/11/2025

Practical 3 - User Space

Command Implementation:

This execution process of the `ls` command follows a two-stage design: a custom system call and a dedicated OS device responsible for computation. The command validates input (path and flags), then invokes the `ls_syscall`, which calls the `ls-device` to initiate a VFS path lookup. The device locates the target directory node and populates an internal buffer with data for all directory entries. After the system call returns, the command opens the device as a file and copies the buffered data back to user space for presentation.

The `ls` technical implementation incorporates a temporal caching mechanism via an `ls-cache` that wraps over an AVL tree map provided by the OS Standard Library. Paths are hashed and associated with their corresponding directory entry metadata. Lookup employs two cache layers to maximize performance:

1. **Last Path Reuse:** If the requested path matches the path from the previous invocation, the internal buffer is retained rather than cleared, enabling direct reuse without an additional write-back. This is achieved by storing the last path and comparing it upon function entry.
2. **Path Hash Cache:** After populating the internal buffer, the resulting struct is registered in the cache keyed by the path hash. Subsequent lookups for a cached path can copy directly from the cache without re-iterating over the directory node's children.

In both cases, a path lookup is performed to validate cache freshness. Each directory node maintains a dirty bit, toggled by the `mkdir` operation. If the dirty bit is set, the implementation iterates the children to repopulate the cache entry before serving the request.

Assumptions:

- **Directory mutation via shell:** There is no user-accessible mechanism to create regular files from the shell. At present, the directory node's dirty bit is toggled only by the `mkdir` function.
- **Hash Function Variance:** The `string` class's `get_hash()` function provides sufficient dispersion to minimize cache collisions for path keys within the AVL-Tree `ls-cache` map.

Optimisations and Additional Features:

- **Layered Caching**: Two-layer caching cuts redundant work on repeat listings: last-path reuse avoids buffer reinitialisation and re-traversal, and the AVL path-hash cache returns results without re-traversal. Together they reduce syscall/device round-trips and I/O on stable directories, maximising the performance of the command.
- **Additional Command Flags:** The command supports extended flags for output control and sorting: `-a` includes hidden entries; `-h` renders human-readable sizes. For ordering, `-S` sorts by file size and `-N` sorts by file name, allowing users to tailor enumeration to their inspection needs.

Challenges and Solutions:

The initial design targeted a syscall-only implementation, but repeated page faults and KVM internal errors occurred when transferring the buffer from user space to kernel space. On the recommendation of Dr. Tom Spink, the design was revised to use an OS device, which provided the required functionality without triggering the faults. Bruteforcing revealed the root cause: a struct consuming approximately 67

KB of stack memory, exceeding the kernel's static stack allowance. After reducing the struct size and related allocations, the faults were resolved and the submission finalized. While the device could now be removed, its presence simplifies the caching implementation, so it remains as a pragmatic choice rather than a strict necessity.