CS3104 P3 – Userspace - 220012756

-The implementation of the ‘ls’ command required modifications across both kernel space and user space. The main challenge was allowing user-space programs to access directory content via kernel space system calls. I chose to implement a dedicated system call (‘readdir’) rather than overloading existing calls, maintaining modularity, keeping the code clean, and allowing for future extensions.

-The do_readdir system call (in syscall.cpp) takes a directory path, a user-provided buffer, and a maximum directory entry value. It retrieves the directory node from the VFS, casts it to a fat_node, and loads its children. The ensure_loaded() method in the fat_node class ensures that directory contents are loaded from disk before reading, preventing segmentation faults due to uninitialised data.

-Each directory entry is represented by a dirent structure containing the filename, type, and size. The kernel verifies that the user-provided buffer is writable before computing an offset into the region’s storage which ensures safe communication between user and kernel space.

-To avoid page faults caused by stack allocation, the ‘ls’ program allocates an array of dirent structures on the heap. This uses the kernel’s memory allocator which maps virtual pages on demand and prevents stack overflow issues.

-The ls program handles three argument cases: no arguments (ls), one argument (either -l or a directory path), and two arguments (-l followed by a path). The long listing flag displays file type and size (if a file), with output spacing of one tab. which is not perfectly aligned with other outputs but is still clear and readable.

-I implemented ‘automatic path resolution’ in the shell for /usr commands. Commands without a leading slash are automatically prefixed with ‘/usr/’, which allows users to type ‘ls’ instead of ‘/usr/ls’. Also, the output is sorted alphabetically with a bubble sort algorithm. I chose this as it's simple to implement, and time complexity O(n^2) is trivial as there can only be a maximum of 256 entries to sort so bubble sort avoids unnecessary over-engineering.

-The decision to use a region-based memory copy approach adds some complexity but ensures safe and correct operation across user and kernel space. Direct copying to user addresses wouldn’t work as the kernel operates in a different address space, so computing the offset into the region's storage was necessary. This highlights the importance of properly managing the memory boundaries between the user and kernel space.

-The main challenge was transferring data from kernel to user space, especially understanding how user virtual addresses map to kernel accessible memory. Another challenge was page faults from allocating large buffers on the stack, which took a long time to track down and was resolved by switching to heap allocation. Current limitations include imperfect spacing/alignment in the output. While it is functional, it could be standardised for clearer output. For assumptions, I am assuming a maximum file length of 256 chars, and a maximum of 256 directory entries.