## Overview of Work Accomplished

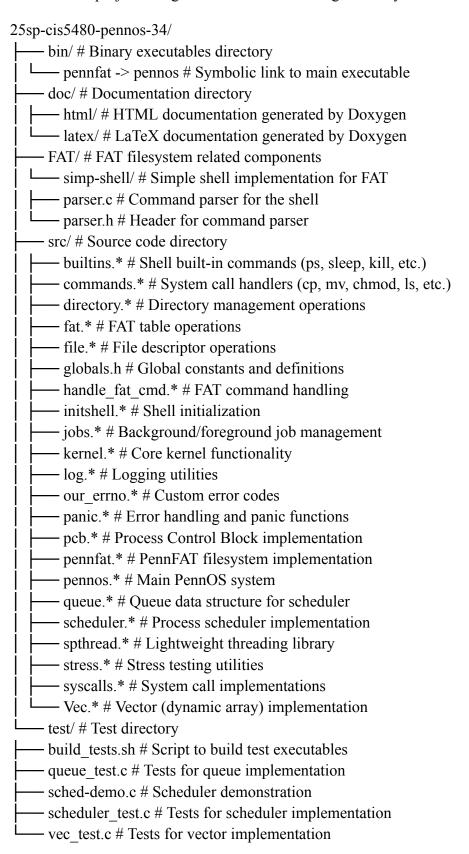
- Implemented a lightweight FAT filesystem (PennFAT) with file and directory operations.
- Designed and implemented a shell capable of running built-in and user commands.
- Built a scheduler supporting multi-level feedback queues and aging for fair process management.
- Implemented custom system calls (k open, k read, k write, k close, k lseek, etc.).
- Added a lightweight user-space threading library (spthread) with manual suspend/resume control.
- Implemented various stress test commands to simulate load and validate scheduler and process robustness.

## Description of Code and Code Layout

- src/builtins.\* Shell built-in command implementations (e.g., ps, sleep, kill, nice).
- src/commands.\* System call and filesystem operation handlers (cp, mv, chmod, ls, etc.).
- src/directory.\* Directory management, creation, lookup, and deletion.
- src/fat.\* FAT table initialization, reading, writing, block allocation, and deallocation.
- src/file.\* File descriptor operations and GFDT (Global File Descriptor Table) management.
- src/kernel.\* Core kernel system call routing and process creation support.
- src/jobs.\* Background and foreground job tracking.
- src/pennfat.\* Filesystem mounting/unmounting logic.
- src/queue.\* Linked list queues used for scheduling processes.
- src/scheduler.\* Scheduler with multiple queues and current running process tracking.
- src/spthread.\* Lightweight thread abstraction using POSIX threading with signal control.
- src/syscalls.\* Wrappers around system calls and user-to-kernel space interactions.
- src/initshell.\* Shell and init process startup sequence.
- src/stress.\* Stress testing utilities to validate robustness under load.
- src/Vec.\* Dynamic array (vector) implementation used internally across modules.

The overall project builds into a shell executable that interacts with the FAT filesystem and manages processes and jobs using a custom kernel and scheduler.

The PennOS project is organized into the following directory structure:



Each component is designed to be modular and focused on a specific aspect of the operating system:

- Core OS Components: kernel, scheduler, PCB, syscalls
- Data Structures: Vec (vector), queue
- Filesystem: pennfat, directory, file, fat
- Shell Interface: builtins, commands, initshell, jobs
- Utilities: log, panic, our errno, stress

## General Comments and Grading Notes

- We have tested file operations (touch, cat, mv, rm, chmod) under different edge cases (permissions denied, missing files, etc.).
- Stress testing functions were used to validate scheduler correctness under high load conditions.
- All built-in shell commands were manually verified in both normal and edge case scenarios.
- Shell prevents termination via Ctrl+C and Ctrl+Z to protect its control over foreground/background processes.
- Doxygen documentation for all structs, functions, and files has been generated and included (refman.pdf).