Operating System
Lab Assignment

[CSL301]
File Systems and Process Control
Time: 90 min

Instructor: Dr. Dhiman Saha

November 19, 2025

Q1: The "Double-Tap" Exit (Two-Strike Signal)

Problem Description

Implement a safety feature where a running user process (opted into this mode) is only terminated upon the **second** consecutive press of `Ctrl+C`. The first press should only issue a warning.

Prerequisites: Wire up the `twostrike` system call.

Kernel Logic: Implement the interrupt handling in `console.c`.

User Test: Create a test program that enables the mode and spins.

Add the System Call: Create a new system call int twostrike(int enabled), Update syscall.h, syscall.c, user.h, and usys.S.

Q1: Task 1 & 2 Setup (`proc.h` / `syscall` / `console.c`)

Task List: Prerequisites and Data Setup

Process Metadata: Update kernel/proc.h by adding two integer fields to the `struct proc` to track the mode and the strike count.

Initialization: Ensure these new fields are initialized to 0 in kernel/proc.c inside the `allocproc` function.

System Call (Syscall) Setup: Implement the `int twostrike(int enabled)` system call in kernel/sysproc.c to set the current process's `twostrike_mode` flag.

REPLACE FUNCTION: You must replace the existing `consoleintr` function in kernel/console.c with the incomplete code provided in kernel_code_q1.pdf.

LOCATE: Find **BLANK 1** through **BLANK 5** in the new `consoleintr` function.

COMPLETE: Fill the blanks to correctly handle the two-strike logic.

Q1: Task 3 User Test Program (`twostriketest.c`)

Task List: User Program

Create File: Create a new user program file: user/twostriketest.c.

Copy Code: Copy the incomplete code provided in user_code_q1.pdf into this new file.

Complete Code: Fill in **BLANK 6** and **BLANK 7** to enable the mode and create a spinning loop.

Makefile: Update Makefile to add `twostriketest` to the `UPROGS` list.

Hints for `twostriketest.c` Blanks:

(B6): Use the name of the system call you defined in Task 3.

(B7): The condition for a spin loop is the simplest value that is always true (e.g., 1).

Q2: Implement a Simplified `ls` Command

Problem Description

You are given an incomplete C program that mimics the basic behaviour of the Linux `ls` command.

- List all files in a directory.
- When called with the `-l` flag, print detailed information about each file, similar to `ls -l`.

Q3: Combined Copy/Move File Utility

Problem Description

You are given an incomplete program that can behave as both a copy (`cp`) and move (`mv`) command depending on how it is executed.

- Perform a copy operation if run as `cp`.
- Perform a move operation if run as `mv`.
- Work only using system calls — `open()`, `read()`, `write()`, `close()`, `stat()`, `rename()`, and `unlink()`.
- Preserve file permissions when creating the destination file.
- Support two usage modes:
  1. By running the executable directly as `cp` or `mv` (e.g., `./cp file1 file2`).
  2. By passing `cp` or `mv` as the first argument (e.g., `./a.out cp file1 file2`).

Q4: File Creation and Deletion with Metadata

Problem Description

You are given a simulation of a simple file system that uses inodes to represent files. Each inode contains metadata such as creator ID, creation time, and a short description.

- Complete the functions to create and delete files.
- Properly fill inode metadata when creating a file.
- Free allocated resources (inode and data blocks) when deleting a file.
- Display correct inode information (creator ID, timestamp, description).

Thank
You