Assignment 2

Developing a Loadable Kernel Module

Submission Deadline: October 5, 2025 EOD

Submission Instructions

1. You need to submit the assignment through CSE Moodle.
2. Only one member from each group should submit the assignment solution as a single zip file.
3. Mention the name and roll numbers of the group members in your submission.
4. Please document your implementation properly. You should include a README file explaining how to compile and run your code, as well as the test cases that you have tested.

Objective

This assignment aims to develop a basic loadable kernel module (LKM) for doing various jobs inside the kernel space.

Please download the 64-bit Ubuntu 22.04 LTS Desktop image and use it in a Virtual Machine (VM).

For tasks involving kernel configuration and building, use kernel version 5.10.240 or something close (5.10.xxx). (You can download kernel code from **kernel.org**)

Task Description

In this assignment, you need to write a **Loadable Kernel Module (LKM)** that provides the functionality of a **Queue** of maximum capacity ≤ N inside the kernel mode. Your LKM should be able to handle **32-bit integers**. Upon insertion of this LKM, it will create a file at the path: /proc/lkm_<roll nos>. This file will be world-readable and writable. A user-space program will interact with the LKM through this file.

Interaction Specification

A user-space process can interact with the LKM in the following manner only:

1. Open the file (**/proc/lkm_<roll nos>**) in **read-write** mode.
2. Initialize the Queue:
   - Write one byte of data to the file to initialize the queue.
   - The first byte should contain the maximum capacity **N** of the queue.
   - The size **N** should be between **1 and 100** (inclusive).
   - If N is not within this range, produce an **EINVAL** error, and the LKM remains uninitialized.

3. Enqueue Operation (Write Calls):
    - Subsequent **write()** system calls will insert one 32-bit integer at a time into the queue (enqueue)
    - On success, the LKM will return the number of bytes written (4 bytes for a 32-bit integer).
    - If the queue is full, the LKM should produce an **-EACCES** error and reject the write.
    - If the argument type/size is invalid, return **-EINVAL**.
4. Dequeue Operation (Read Calls):
    - A **read()** call will remove and return all the elements currently in the queue in FIFO order.
    - On success, the number of bytes read = (#elements × 4).
    - If the queue is empty, return **-EACCES**.
    - read() calls are also made between write calls and state of the queue is verified for consistency.
5. Close:
    - The user-space process should close the file once done.
    - On closing, the LKM must free up any resources allocated for that process.

Additional Requirements

1. The LKM should be able to handle concurrency and maintain separate queues for multiple processes.
2. No user-space program should be able to open the file more than once simultaneously.
3. A process may reset its queue by closing and reopening the file.
4. Proper memory cleanup must be ensured when the process closes the file.

Testing

You should test your LKM implementation with multiple user-space processes performing enqueue and dequeue operations simultaneously. Verify that:

1. Each process has its own private queue.
2. Concurrency is handled correctly.
3. Invalid operations produce the correct error codes.
4. The state of the queue for each process is consistent after the operations being done (Make similar queues (using C++ stl library) in user space and verify)