Project Report

MUTEX (Multi-User Threaded Exchange Xfer)

A Kernel-Level Proxy Service Module for Linux

Course: CS 311 Operating Systems

Date: December 21, 2025

Team Members

Azeem (2023150)

Hamza Bin Aamir (2023219)

Syed Areeb Zaheer (2023672)

Introduction to the Problem

Traditional proxy services and VPNs typically operate at the **Userspace** level. When an application sends data, the packets must travel from the kernel to the userspace proxy application, be processed, and then be sent back down to the kernel to be transmitted over the network. This context switching between User Mode and Kernel Mode introduces significant overhead, increased latency, and reduced throughput.

Furthermore, user-level proxies often require manual configuration for every application or complex routing table manipulation that are difficult to manage.

The Windows vs. Linux Landscape:

Notably, functionality for kernel-level network interception and filtering (such as the Windows Filtering Platform - WFP) has existed in the Windows Operating System for years, allowing developers to create highly efficient proxy and security drivers. However, such a streamlined, file-descriptor-based transparent proxying mechanism has not been natively implemented in the Linux kernel as a unified feature. Developers are currently forced to use complex combinations of iptables, nftables, and tproxy targets, which are often cumbersome to configure dynamically.

The MUTEX Solution: MUTEX addresses these gaps by implementing a **Loadable Kernel Module (LKM)**. By hooking directly into the Linux network stack using Netfilter, MUTEX intercepts and rewrites packets within kernel space. This approach:

1. **Eliminates Context Switching:** Data stays in kernel space, maximizing performance.
2. **Provides Transparency:** Applications do not need to be aware of the proxy; the kernel handles the redirection automatically.
3. **Unified Control:** Uses a standard Unix "Everything is a File" paradigm to manage proxy states.

The Unique Function: mprox_create()

The core innovation of the MUTEX project is the custom system call mprox_create(). Unlike standard networking tools that rely on global configuration files, MUTEX allows per-process or per-session proxy control through a file descriptor.

Function Signature

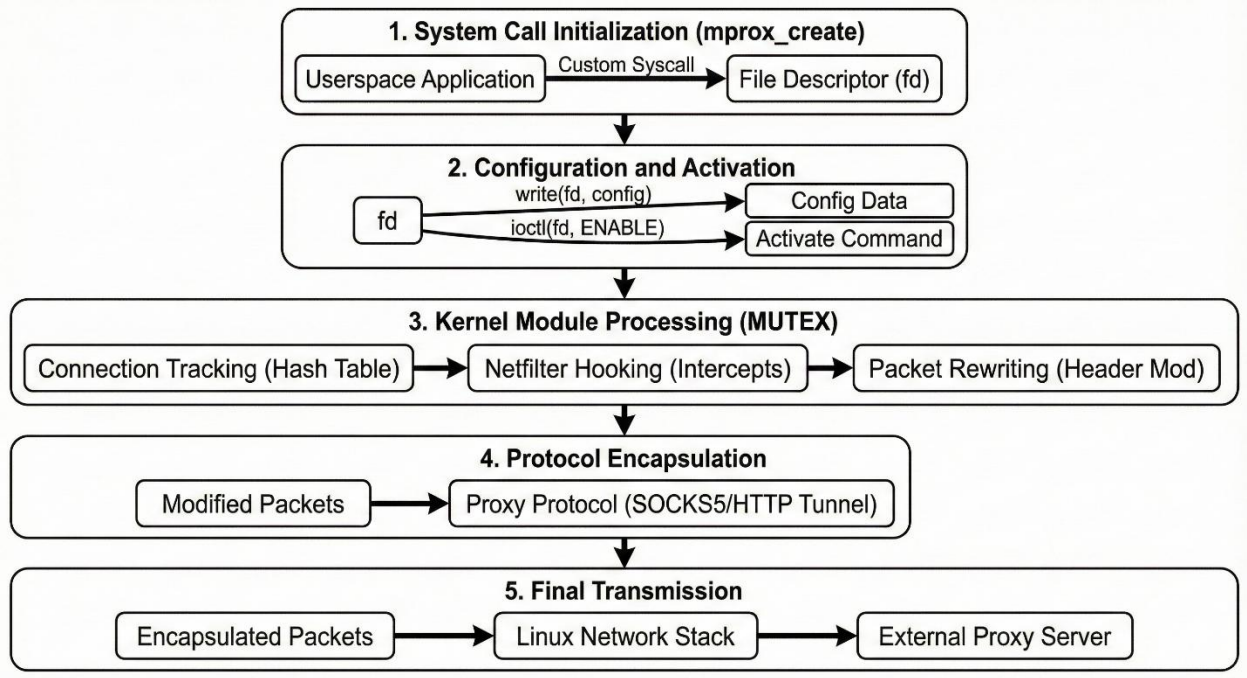
int mprox_create(unsigned int flags);

Unique Features:

- **Anonymous Inode Design:** Similar to eventfd() or timerfd(), this syscall returns a file descriptor associated with an anonymous inode in the kernel.
- **File Descriptor Control:** Once a process obtains this fd, it can configure proxy settings using standard write() operations and control the proxy state via ioctl().
- **Inheritance:** Because it is a standard file descriptor, it can be inherited by child processes during a fork(), allowing a parent (like a shell or a daemon) to enforce proxy rules on its entire process tree.
- **Capability Check:** The function performs strict CAP_NET_ADMIN checks to ensure only authorized users can initialize the proxy service.

4. System and Function Call Diagram

The following diagram illustrates the flow from a userspace application through our custom syscall and into the kernel networking hooks.



5. Execution Commands

To build, load, and execute the MUTEX project, follow these steps:

A. Build the Kernel Module

Navigate to the module source directory and compile:

```
cd src/module
make
```

B. Load the Module

Insert the compiled module into the Linux kernel:

```
# Requires root privileges
sudo insmod mutex_proxy.ko

# Verify the module is loaded
lsmod | grep mutex_proxy

# Check kernel logs for successful initialization
dmesg | tail -n 10
```

Not necessary if module compiled into the kernel.

C. Build Userspace Tools & Library

Build the libmutex library and the mprox CLI tool:

```
cd ../userspace
make
```

D. Execute the Proxy Controller (CLI Tool)

The mprox CLI tool is the primary interface for managing the proxy. It utilizes the libmutex library to communicate with the kernel module.

Usage Examples:

Option 1: Using Example Programs (Recommended for Testing)

Simple proxy example:

```
cd /home/areeb/MUTEX/src/userspace
sudo LD LIBRARY_PATH=./lib/examples/simple_proxy
```

This demonstrates the complete workflow: create > configure > enable > use

Multi-FD example (demonstrates multiple proxy contexts):

```
sudo LD_LIBRARY_PATH=./lib/examples/multi_fd
```

Poll example (demonstrates event handling):

```
sudo LD LIBRARY PATH=./lib/examples/poll_example
```

Check Version:

```
LD_LIBRARY_PATH=./lib ./cli/mprox version
```

View Statistics:

```
# Displays real-time bandwidth and connection statistics from the kernel
LD_LIBRARY_PATH=./lib ./cli/mprox stats
```

Run with a Configuration File:

```
# Applies settings from a JSON configuration file
LD_LIBRARY_PATH=./lib ./cli/mprox config --file ../config/default.json
```

E. Unload the Module

When finished, remove the module from the kernel, if loaded via insmod:

```
sudo rmmod mutex_proxy
```