VirtualMouse Design and Specifications

V1.0

Prepared by Edwin Heerschap

May 7, 2020

Contents

Preface

VirtualMouse is a linux kernel driver aimed at providing programmatic mouse functionality. It is encompassed by the VirtualHideout project by SneakyHideout; which aims to create virtual peripherals. This document describes the software design and specifications for VirtualMouse v1.0. This document is not developer documentation for parts of the project.

Software Design Overview

Abstraction

A seperation of concerns approach is taken for the design of VirtualMouse. This materializes in a layering pattern for the system. Four primary layers are present in the highest abstraction of the system; kernel driver, interface library, userspace implementation, other library. Interactions among them are represented in figure 1.

Kernel Driver

VirtualMouse kernel driver is the core of the virtual mouse system. It is a character device driver for the linux kernel. Systems interpreting mouse protocols such as X11 or other kernel drivers will read input from here. The interface library requests actions from the kernel driver resulting in mouse events.

Interface Library

The interface library is a native shared object file with methods to interact with the kernel driver. Maintaining an interface library removes userspace dependence on the kernel driver implementation. Secondly, it allows language independent access to kernel driver functionality.

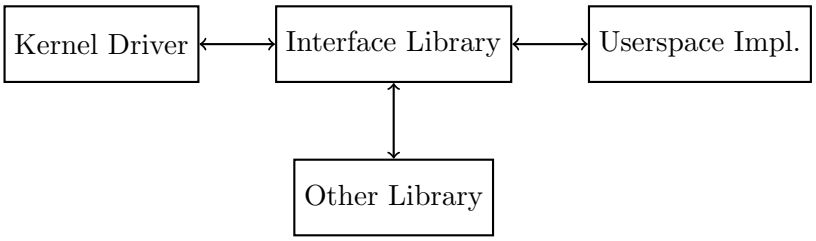
Userspace Implementation

Userspace implementation is userspace software employing the interface library to create mouse events.

Other Library

Positional information of virtual mice is not stored in the kernel driver. It is the responsibility of software such as X11 to maintain positional information. Other unknown information as of current may also be unavailable to the kernel driver. Other libraries will get this information.

Figure 1: Interaction between abstractions



Kernel Driver

The design of the kernel driver will attempt to comply with the S.O.L.I.D design principles[1] as closely as realistically possible (as *C* is not directly object oriented).

In this description the semantics for a structure having a method is a function exists which accepts the structure as one of its parameters. For example, *vmDevice*'s ioctl() method accepts a (*struct vmDevice**) as one of its parameters.

Specifications

The kernel driver is required to support multiple virtual mice. They may also be running concurrently. Each mice may have their own:

- Protocol
- Name
- Access control (Locking)
- Input/Output Control (IOCTL) access

Furthermore it is required that mice can be added and removed during runtime. They may also be specificed as input parameters. To be more verbose, the kernel driver must be capable of:

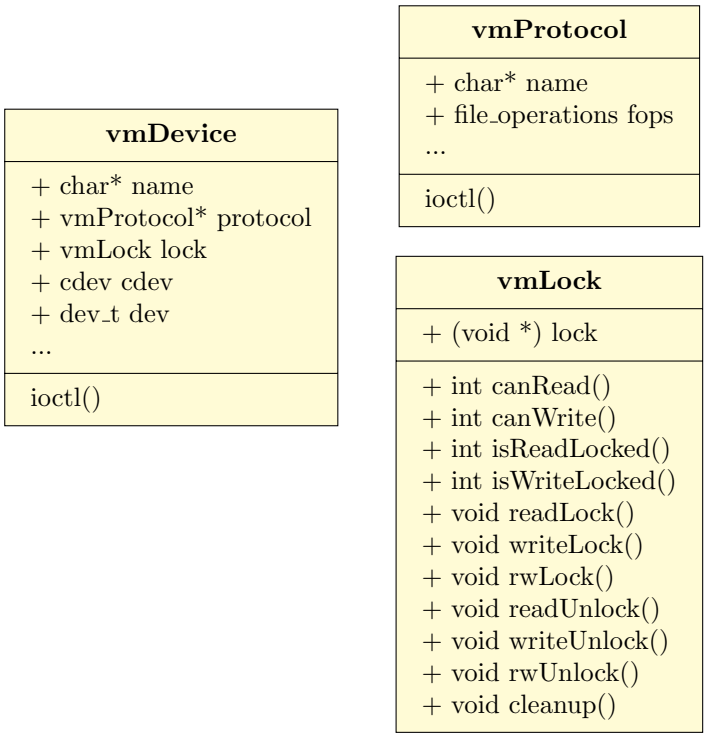
- Mice
  - Select the protocol
  - Select the name
  - Select type of access control

[1] https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

    - Perform mice actions
- Add new mice during runtime
- Remove mice during runtime
- Add mice as startup parameters

Protocol and syntax on how to achieve each of these will be specified further into the initial development.

Figure 2: vmDevice structure & dependencies



vmDevice structure

Figure 2 is a UML representation of the general software structure of *vmDevice*. *vmDevice* is an individual virtual mouse. A *vmDevice*'s protocol is stored as a pointer as multiple mouse can use a single protocol. However, a *cdev, dev, lock* is exclusive to the device and hence not pointers. *vmDevice*'s structure is used to store its state and point to behavior. The ioctl() method mechanics is described in the IOCTL Chain section.

vmProtocol represents a mouse protocol; the bus protocol for example. System interactions following protocol are defined in *fops*. Non-ioctl device interactions are routed to the *fops* methods unless blocked by the fops proxy; by a lock for example. The ioctl() mechanics are described in IOCTL Chain section.

vmLock represents the access control (concurrency management) system used by the device. Each of its methods responses are respective to the *lock* attribute.

IOCTL Chain

file_operations proxy