Universidade de Brasília

Engenharia de Software

Laboratório 3 (*Device Drivers* e Vírus)

Autor(es):
Paulo Tada

Professor(es):
Fernando Willian Cruz

July 4, 2016

1 Introdução

Este projeto tem como objetivo a criação de um *device driver* para a plataforma Linux e um vírus específico para este *driver*, além de realizar estudo sobre os dois temas.

1.1 *Device Driver*

Um *device driver* é um mecanismo especial do Linux kernel que permite ao programador construir uma lógica através de uma interface de comunicação com um dispositivo de IO (*Input and Output*). Com essa interface o *driver* pode ser construído a parte do resto do kernel e "plugado" em tempo real quando necessário.

Na construção de um *device driver*, é importante esclarecer dois conceitos:

- ***Kernel Space***: o kernel gerencia o *hardware* de maneira simples, oferecendo ao usuário um simples e uniforme modo de programação, as interfaces. Sendo assim, o kernel promove uma ponte entre o usuário/programador e o *hardware*. Qualquer subrotina e função que façam parte do kernel são considerados parte do *kernel space*.

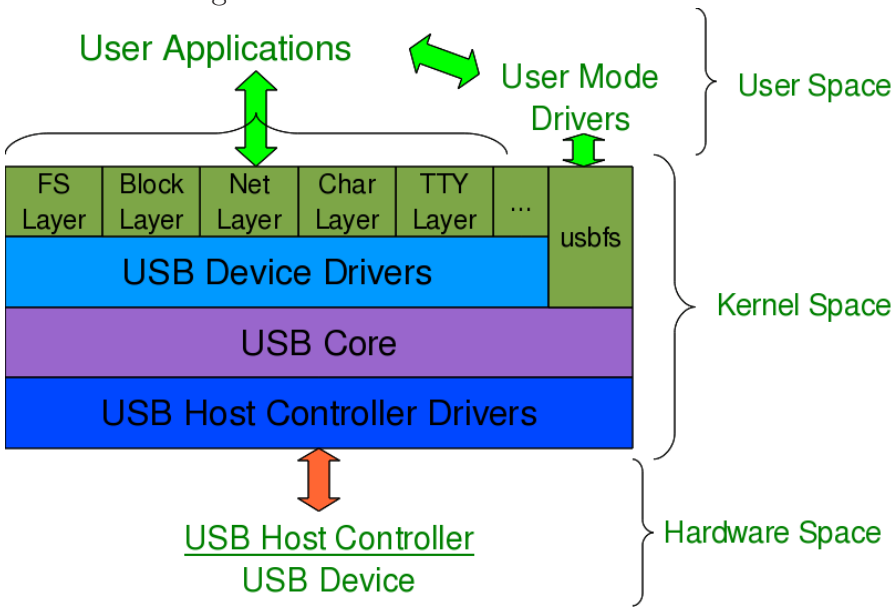
- ***User Space***: os programas *end-user* como o UNIX shell ou outra aplicação GUI são parte do *user space*. Elas se comunicam com o *hardware* do sistema, mas não diretamente. O kernel promove funções de suporte para essas aplicações e o próprio kernel realiza a gerência do *hardware*.

Em resumo, kernel oferece subrotinas ou funções para as aplicações *end-user* para interagir com o *hardware* (user space), mas também oferece funções para a comunicação baixo nível do sistema com o *hardware* (kernel space).

1.1.1 USB Driver

Em um sistema operacional Linux, um dispositivo USB sempre será detectado quando inserido. A detecção em nível de *hardware* é feito pela controladora de USB - *USB Host Controller*. A controladora correspondênte irá enviar as informações da camada *low-level* para a camada *higher-level* que contém as especificações do protocolo USB. As especificações de protocolo transformam as infomações recebidas sobre o dispositivo e os propaga para a camada de USB genérica - *USB Core* - definida no espaço do kernel. Nesse ponto, o device é detectado mesmo sem haver um *driver* associado específicamente a ele.

Figure 1: Camadas do sistema USB Linux



A camada de USB core se comunica com o dispositivo utilizando os *urb* que é descrito com uma estrutura (*struct urb*). Os urbs são utilizado para enviar e receber dados de um USB endpoint de um dispositivo específico. O ciclo de vida de um urb é basicamente:

1. Criado por um driver;
2. Associado a um endpoint de um dispositivo USB;
3. Submetido para o USB code pelo driver;
4. Submetido para o USB host controller específico do dispositivo;
5. Processado pelo USB host controller que transfere para o dispositivo;
6. Quando o urb é completado, o USB host controller notifica o driver.

Um urb é criado dinamicamente e pode ser concelado a qualquer momento pelo driver ou pelo USB core se o dispositivo for removido do sistema.

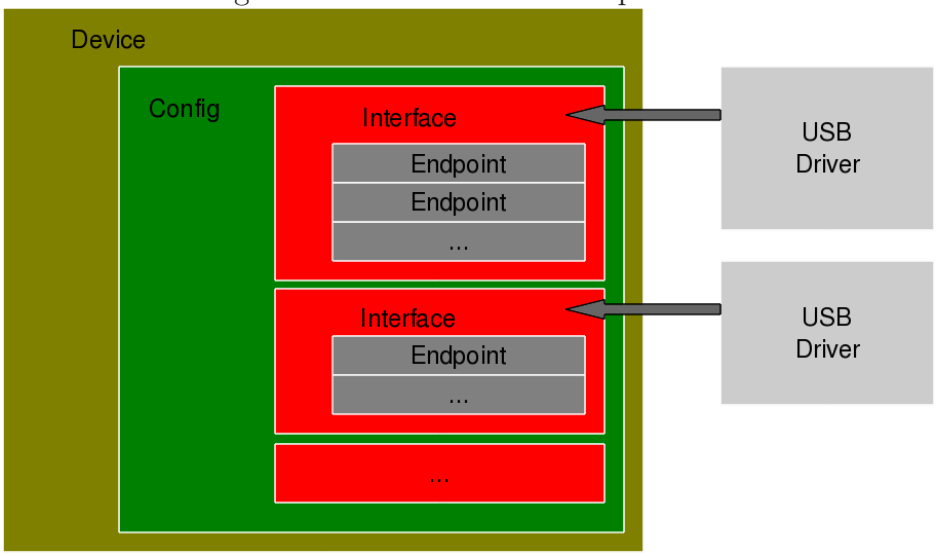
Um dispositívo válido de USB tem uma estrutura composta por: *configurations, interface e endpoints*.

- **Configurations**: é o perfil do dispositivo. É usado pelo USB Core para inicializar o dispositivo;
- **Interface**: corresponde a funcionalidade do dispositívo;

- **Endpoint**: são como *pipes* que transferem informações para o dispositívo ou para o computador.

O ambiente Linux suporta apenas uma *configuration* por dispositivo. Para cada *configuration*, o dispositivo pode tem uma ou mais *interface*. Como dito, uma *interface* representa uma funcionalidade do dispositivo, por exemplo, uma impressora multifuncional tem as funções de imprimir, scanner e fax. Portanto, um dispositivo USB, ao contrário de outros dispositivos, é associado por *interface* ao invés do dispositivo como um todo. Em resumo, um dispositivo USB pode tem vários drivers e várias *interfaces* podem ter o mesmo driver. E para cada *interface* tem-se vários *endpoints*. A figura **??** mostra essa estrutura.

Figure 2: Estrutura de um dispositivo.



Na seção **??** será mostrado algumas aplicações e comandos que são utilizados para identificar as informações de um dispositivo USB.

1.2 Vírus

- **Vírus de *Boot***: o vírus de boot foi um dos primeiros vírus a surgirem no mundo, ele foi criado com o objetivo de danificar o setor de inicialização do disco rígido, no qual a sua forma de propagação era através de um disquete contaminado. O vírus se alojava no primeiro setor do disco flexível e ocupava cerca de 1k ou menos em um disquete que era de 360k mais ou menos. Na inicialização do boot com o disquete contaminado, o vírus se aloja na memória no endereço 0000:7C00h da

bios e começa a se auto-executar , prejudicando assim a inicialização do computador

- **Vírus de Macro**: o vírus macro encontrado-se frequentemente em documentos ou é inserido em códigos maliciosos em programas de processamento de texto, podendo ter origem em docs anexados ou em e-mails, no qual o vírus pode ser transferido para o computador depois de clicar em ligações de "phishing" ou mesmo em propagandas. Esses vírus são difíceis de serem identificados e o seu maior risco é que tem uma grande capacidade de se propagar, no qual eles podem causar danificações nos documentos infectados.

2 Descrição do Problema

Dentro do contexto de *device drivers*, existe o aspecto de segurança do sistema operacional. Ataques externos são considerados qualquer tentativa de recuperar dados sigilosos, danificar o sistema em sí ou utilizar recursos computacionais para realizar outras operações não autorizadas. Com visão deste cenário, será necessário construír um vírus para um *driver* específico. No caso deste projeto, o vírus será direcionado para um *driver* construído especificamente para este objetivo, contendo dois comportamentos: normal e anormal, para apresentar o funcionamento do vírus.

3 Metodologia

O grupo adotou reuniões presenciais e remotas para desenvolvimento do trabalho. A partir disso algumas ferramentas foram utilizadas para que se pudesse desenvolver o trabalho em equipe. As ferramentas foram:

- Google hangout;
- Tmate;
- Github;

O Google hangout fora utilizado em conjunto com o tmate para provimento de reuniões remotas e compartilhamento do terminal respectivamente. O tmate com seu compartilhamento de terminal via ssh permite todos integrantes tenham acesso á um terminal único em tempo real. Desse modo, o pareamento remoto torna-se facilitado. O Github é utilizado como ferramento de versionamento de código.

4 *Checklist* de Requisitos

A lista abaixo contem os requisitos especificados para o laboratório e o seu status de implementação da solução proposta.

- ( ) Estudo sobre Vírus: tipos de vírus e vermes, como eles se disseminam;
- (OK) Estudo sobre *Device Drivers*: funcionamento e como construir;
- (OK) Tutorial da construção do *Device Drivers* (Anexo A);
- ( ) Propor um *Device Drive* com duas forma de funcionamento: normal e anormal;
- (OK) O *Device Drive* deve ser gerenciado por lsmdo, insmod, rmmod;
- (OK) O *Device Drive* deve apresentar na tela o que está ocorrendo;
- (OK) Usar processos em ambiente Linux/Linguagem C;
- ( ) Executar o programa várias vezes e criar um quadro adequado para apresetar os resultados;

5 Descrição da Solução

A solução proposta consta com a leitura de um device driver para um mouse USB.

5.1 Comandos de Suporte

Durante a construção de um *device driver* existem algumas aplicações e comandos que facilitam a obtenção de informações sobre o dispositivo e o *driver*.

5.2 Funções da API

6 Conclusão