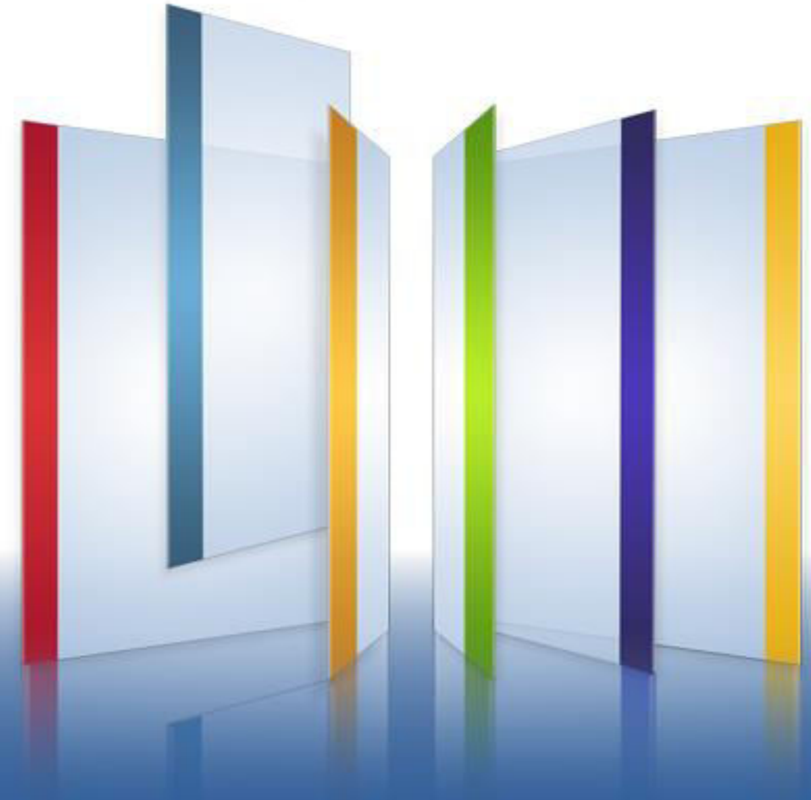
CHECK POINT
INSTITUTE FOR
INFORMATION SECURITY

בית הספר למדעי המחשב על שם בלבטניק
The Blavatnik School of Computer Science

Check Point
SOFTWARE TECHNOLOGIES LTD.



Lecture 2: Linux Devices

Agenda

1 Reminder

2 Linux Devices

3 About next Assignment

Agenda

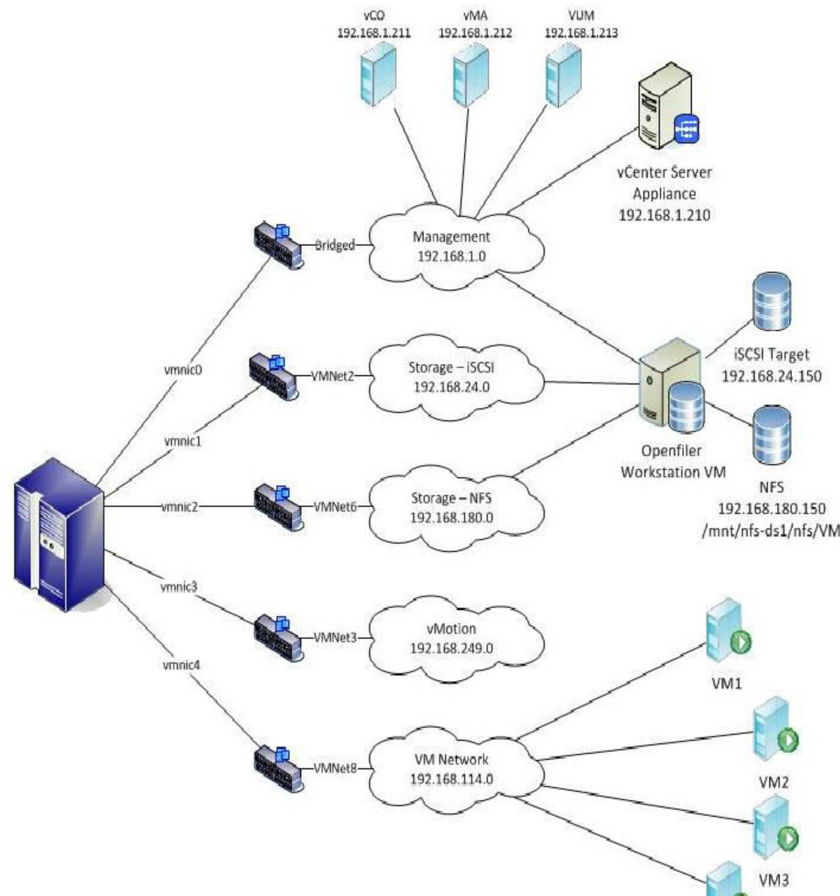
1 **Reminder**

2 Linux Devices

3 About next Assignment

Virtual Network

- You've all experienced setting up virtual lab with machines and networks



What is a Kernel Module

- What is a kernel module?
  - An object file that contains code to extend the running kernel, or so-called base kernel, of an operating system.
- Kernel module can be used as a hardware devise manager, or to add new software features
  - Since we wont use any hardware in this workshop, we ill focus on adding features

Linux Networking

- We need a way to see the packets
- Packets has headers for each layer and data
- Linux contains set of useful tools and structures to manage packets over the kernel

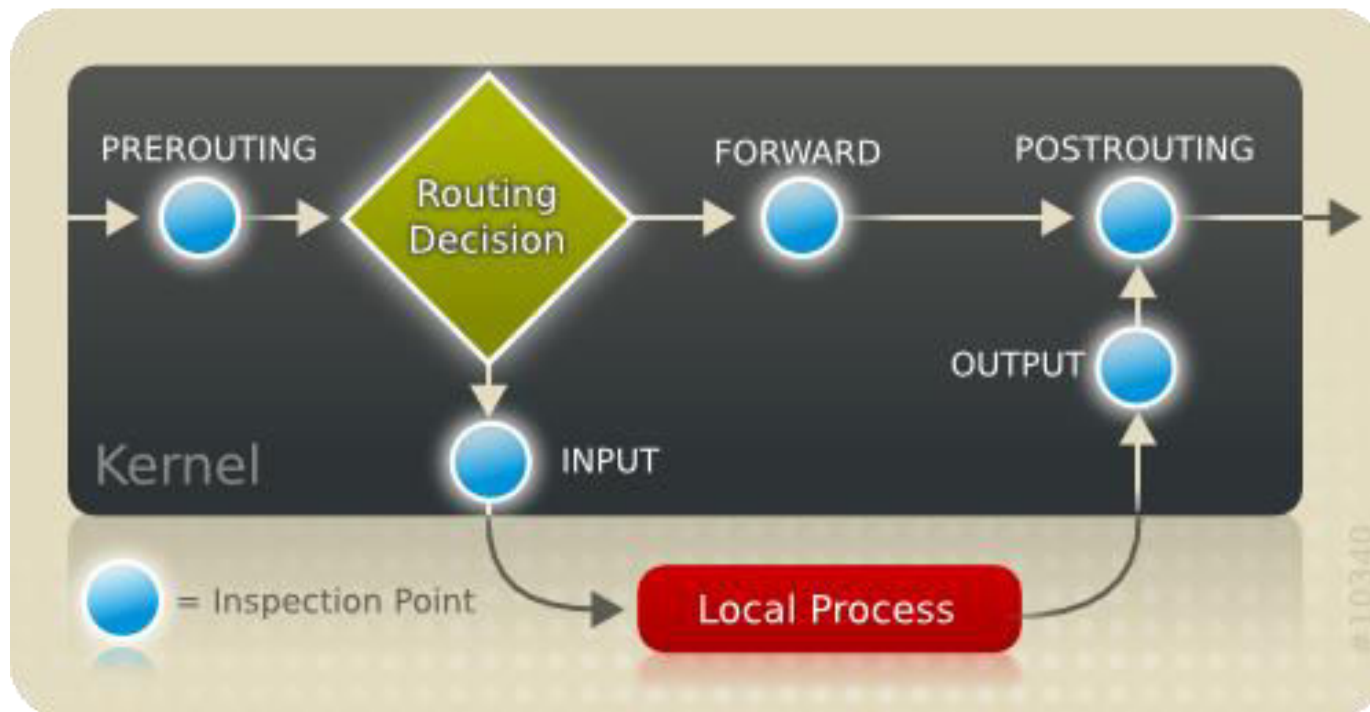
Packet Headers

- Each packet comes with headers, each for every layer.
- The layers we are interested in are the network layer and transport layer
- We can use the information to decide the verdict for each packet\connection

IP header	TCP/UDP header	Data HTTP/1.1 200 OK0x20Last-Modified: Mon, 07 Apr 2014 09:16:25 GMT0x20Content-Type: image/jpeg0x20Cache-Control: max-age = 3153600000x20magicmarker: 10x20Content-Length: 47060x20Accept-Ranges: bytes0x20Date: Mon, 07 Apr 2014 11:35:46 GMT0x20X-Varnish: 2786274250 27578957660x20Age: 83980x20Via: 1.1 varnish0x20Connection: keep-alive0x20X-Cache:

NetFilter

- 5 inspection points which can be used to define behaviors for each type of packets



Our Kernel Module – The Firewall!

- What will we do with our kernel module?
- Register our own functions (AKA: hooks) with the Netfilter API, to issue verdicts on packets going in/out/through our Linux box.

- Register a char device, to communicate with the user space
  - Send commands to set module values.
  - Receive data from the firewall about the state of the system.

Agenda

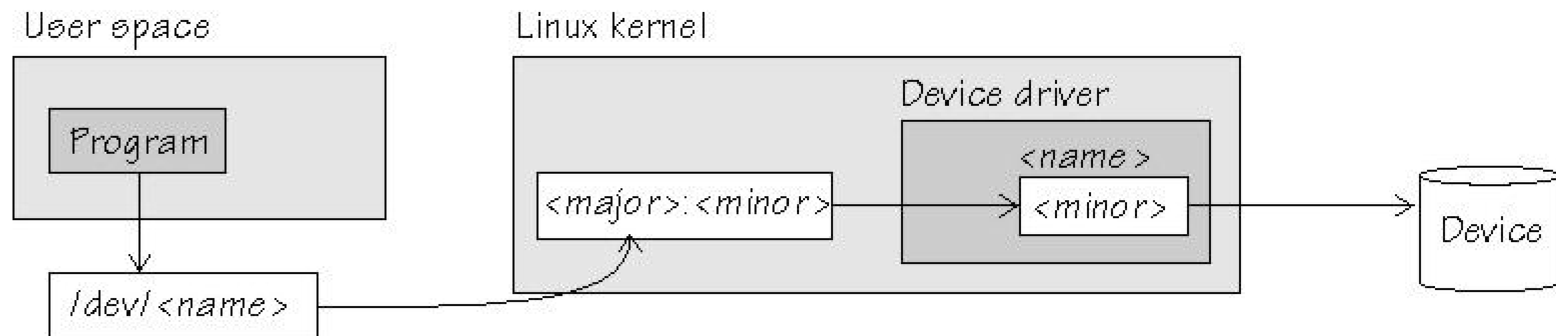
1. Reminder
2. **Linux Devices**
3. About next Assignment

Devices

- There are three kinds of devices in Linux:
  - Character devices – read/write single bytes.
    - Examples: keyboards, mice, printers, and most pseudo-devices.
  - Block devices – read/write blocks of bytes.
    - Examples: hard drives, Blu-ray discs, and memory devices such as flash.
  - Network devices – access to internet via physical adapter
- Now days Linux kernel has a unified model for all devices
  - The device model provides a single mechanism for representing devices and describing their topology
    - For further reading on kobjects, ktypes and ksets click here
    - We deal with more higher level objects
- More information - Linux Kernel Development, page 337.

Character Devices

- We access char devices via names in the file system
  - Located under /dev
- Every char device has its major number and minor number.
  - Major #: unique ID for each device manager
  - Minor #: ID for each device connected to the device manager
- A device can define its own operations on its interface files.



Add new char device

- To create new char device, we need to allocate him new major#
  - The function `register_chrdev()` does it for us
- Syntax:
  - `int register_chrdev (unsigned int major, const char* name, const struct file_operations* fops);`
    - `major` - major device number or 0 for dynamic allocation
    - `minor` - name of this range of devices
    - `fops` - file operations associated with this devices
- Function returns 0 for success or -1 for failure

File Operations

- After registering our char device, it's linked to the major #
  - We will create a representation of it in the file system with `mknod` or the function `device_create`, if used with sysfs
    - Can be seen under /dev/<device_name>
- The "struct file_operations (AKA: fops)" contains mainly pointers to functions.
- It is used to plant our own implementations to various file system calls, like opening or closing a file, and much more.
- First, we define and implement our functions, with the right signature.
- Then, we build an instance of this struct, and use it when we register our char device.

Example

- A scenario:

```
me@ubuntu:~$ ls -l /dev/my_device*
crw-rw-rw- 1 root root 250, 0 Aug 15 12:07 /dev/my_device1
cr--r--r-- 1 root root 250, 1 Aug 15 12:07 /dev/my_device2
me@ubuntu:~$ cat /dev/my_device2
Hello device2's World!
```

- The 'cat' called our implementations of open, read, and release(close).
  - We need to support these functions if it's different from the default operation

```
struct file_operations my_fops = {
        .owner = THIS_MODULE,
        .open = my_open_func,
        .read = my_read_func,
        .release = my_release_func
        };
```

- Owner can be set also through macro: `SET_MODULE_OWNER(&fops);`

Remove char device

- When we want to remove a device from the system (usually in the `cleanup_module` function) we will use `unregister_chrdev()`
- Syntax:
  - `unregister_chrdev(unsigned int major, const char *name)`
    - `major` – the major related to the device we want to remove
    - `name` – the name of the device
- Function returns 0 for success or -1 for failure

sysfs

- A brilliant way to view the devices topology as a file system
- It ties kobjects to directories and files
- Enables users (in userspace) to manipulate variables in the devices
- The sysfs is mounted in /sys
- Our interest is by the high level class description of the devices
  - /sys/<CLASS_NAME>
- We can create devices under this CLASS_NAME
  - /sys/<CLASS_NAME>/<DEVICE_NAME>
  - `/sys/class/my_class/my_first_device/dev_att`

Device Class

- Device class is a concept introduced in recent kernel versions.
- Helps us maintain a logical hierarchy of devices (not to be confused with char devices!)
- Every device has the char-device's major#, and a minor# of its own.

sysfs (cont)

- Just as in read and write, we will have to implement input and output to the sysfs files
- When we will create device files we will have to define device_attributes
  - Pointer to `show` function
  - Pointer to `store` function
- We can just use
  - `echo "whatever" > /sys/<CLASS_NAME>/<DEVICE_NAME>/store`
  - Where is the catch?
  - We can only move data that is smaller than `PAGE_SIZE`
  - A convention is to use human readable data

Sysfs (AKA: /sys)

- To create such file:
  - Create read/write(show/store) functions.
  - Create a Device Attribute for the file.
  - Register the Device Attribute using sysfs API, under your desired device.
- A scenario:

```
me@ubuntu:~$ cat /sys/class/my_class/my_first_device/num_of_eggs
2
me@ubuntu:~$ echo spam > /sys/class/my_class/my_second_device/worm_whole
```

Internal	**External**
Kernel Objects	Directories
Object Attributes	Regular Files
Object Relationships	Symbolic Links

How to work with sysfs

- First, we need to register our device
  - `register_chrdev(unsigned int major, const char *name, struct file_operations *fops);`
    - We will set major to 0 to let the system dynamically give us a major
    - Don't forget `unregister_chrdev` in the `exit` function
- Later, we will create a class to see under /sys/class
  - `struct class * device_class = class_create (struct module *owner, const char *name);`
    - Don't forget `class_destroy` in the `exit` function
  - Owner: the module owner, usually set to the macro `THIS_MODULE`
  - Name: the name you picked for the class
  - The function returns struct class

Continue to set up the environment

- After we created the class folder, we now want to put our device in it. We create device which will link our device into the class
  - `struct device * device_create (struct class * class, struct device * parent, dev_t devt, void * drvdata, const char * fmt, ...);`
    - Don't forget `device_destroy` in the `exit` function
  - Class: the struct class that this device should be registered to, which we created.
  - Parent: pointer to the parent struct device of this new device, if any (usually NULL)
  - Devt: the dev_t for the char device to be added – we need to make a unique number for this device. We have a macro which provide us this number: MKDEV(major number, minor number)
  - Drvdata: the data to be added to the device for callbacks (usually NULL)
  - Fmt - string for the device's name

Continue to set up the environment

- We have now put our device in the class, and we can reach it's default attributes.
- We don't care about the default, and we will usually need to implement 2 new ones (if we need them):
  - Show: show data to the userspace, similar to read
  - Store: send data to the kernel space
- After implementation, we link the device with these attributes with the macro `DEVICE_ATTR(name,mode,show,store)`
- We will use `device_create_file(struct device *device, const struct device_attribute *entry);` to have our new attributes in the device folder
  - Don't forget `device_remove_file` in the `exit` function

References

- Further reference:
  - Linux Device Drivers, Third Edition
    - An excellent free e-book, contains all you need and don't need to know about kernel modules.
    - Written for kernel 2.6, but not a lot changed since.
  - Kernel Headers and Documentation
    - On your machine
      - e.g. /usr/src/linux-headers-`uname -r`/include/linux/ip.h
    - On the net
      - LXR or any other cross-reference site.
      - http://kernel.org/doc/Documentation/
    - The hardest to read, but probably the most useful.
  - Google, Stackoverflow, and Wikipedia has everything you need

Agenda

1 Reminder

2 Linux Devices

3 **About next Assignment**

Assignment 2

- This assignment we will continue to learn about Linux kernel features and APIs which will help us along the way
- We will need to implement connection between your module and a user-space C program
- Use sysfs as you learned in this lecture
- Sysfs is a popular way to connect to the kernel, use the internet to help you figure out what you don’t inderstand
- Make sure your program prints exactly what it should – no more, no less, pay attention to spaces.

Assignment 2

- Every feature from assignment 1 should stay and work in the firewall.
- Assignment 3 will also be published. If you finish assignment 2, I recommend start working on assignment 3 as soon as possible, this assignment is a major leap forward in terms of completion.
- Feel free to contact us for any question.