Design Patterns for modular services, drivers and user land

Bernd Onasch
dieCobol.de GmbH

2016-02-01
MINIXCON 2016
Amsterdam

Abstract

While porting MINIX to specific hardware the requirement of highly modular services and drivers became apparent. We need standardized MINIX micro-kernel compliant interfaces on service and driver level to be able to add new hardware components like network cards or complete new platforms.

A generalized approach of "starting MINIX up" which is as independent as possible from hardware and boot device will additionally help making new hardware supportable.

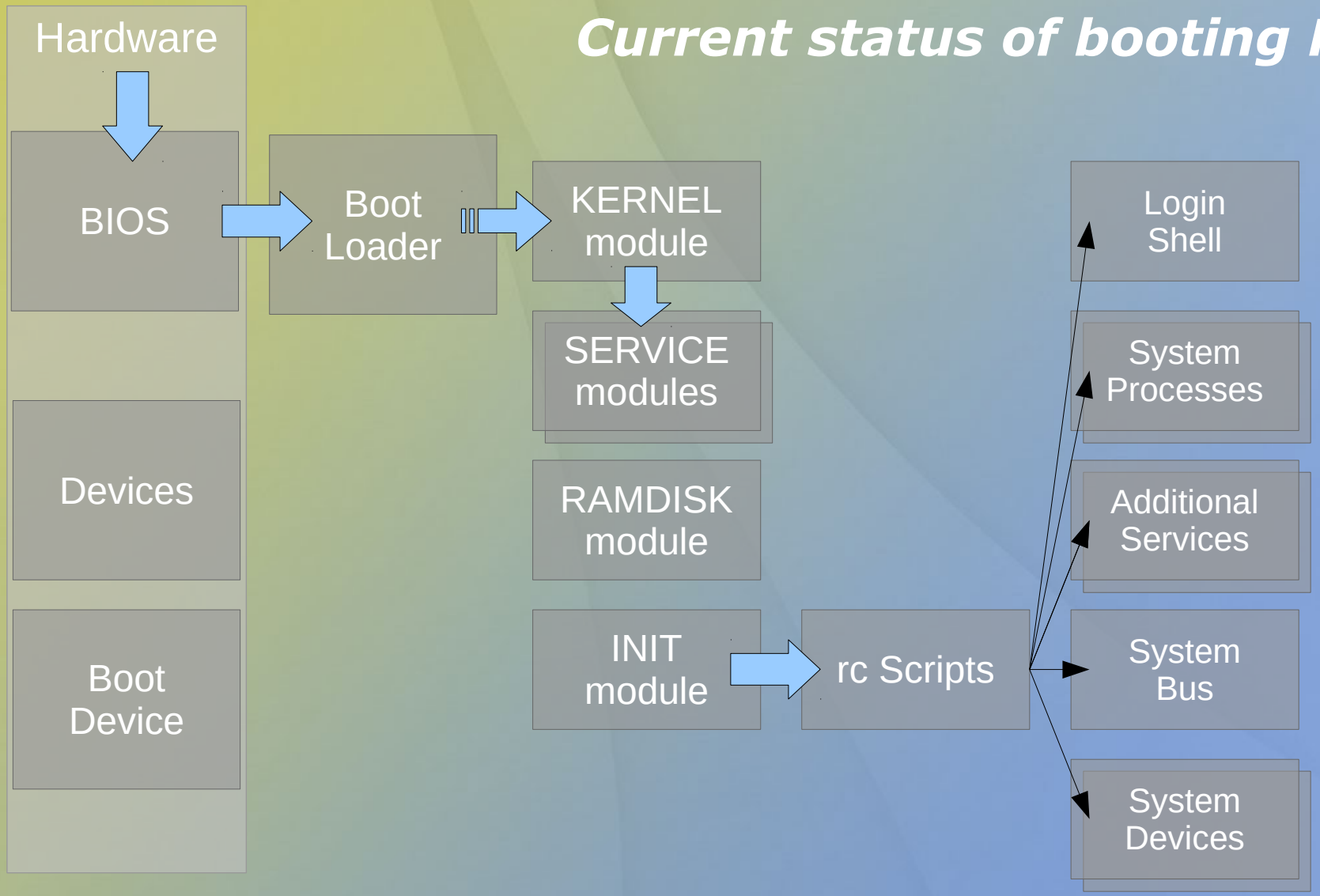
Using a highly modular approach will allow MINIX to reach out to many platforms so far unsupported.

Boot Process

- MINIX boot process (1) … (3)
- Modularized booting? (4)
- Dynamic booting? (5)

MINIX boot process (1)

Current status of booting MINIX



Boot process (2)

We are aiming for a generalized booting process

(1)Hardware and BIOS

- BIOS provides hardware support, e.g. USB
- BIOS initiates boot loader (block device or network)

(2)BIOS and boot loader interaction

- Modern BIOS and boot loader share functionality
- Boot loader is parametrized to know target OS specifics

(3)Boot loader and kernel interaction

- Stage the loaded kernel for 32/64 bit mode and graphics mode

What does the KERNEL really need to start?

How much more does the Developer want to have?

MINIX boot process (3)

Starting MINIX kernel and core services

- Micro-kernel initializing
  - Hand-over from boot loader (multiboot data record)
  - Core service startup (modules loaded by boot loader)
    - Virtual memory, ProcessMgr, Scheduler, Reincarnation, …
  - RAMdisk available to INIT process
- Initial system bus (PCI) validation
  - Initialisation based on pre-defined RAMdisk scripts

„Hen-and-egg”-Problem

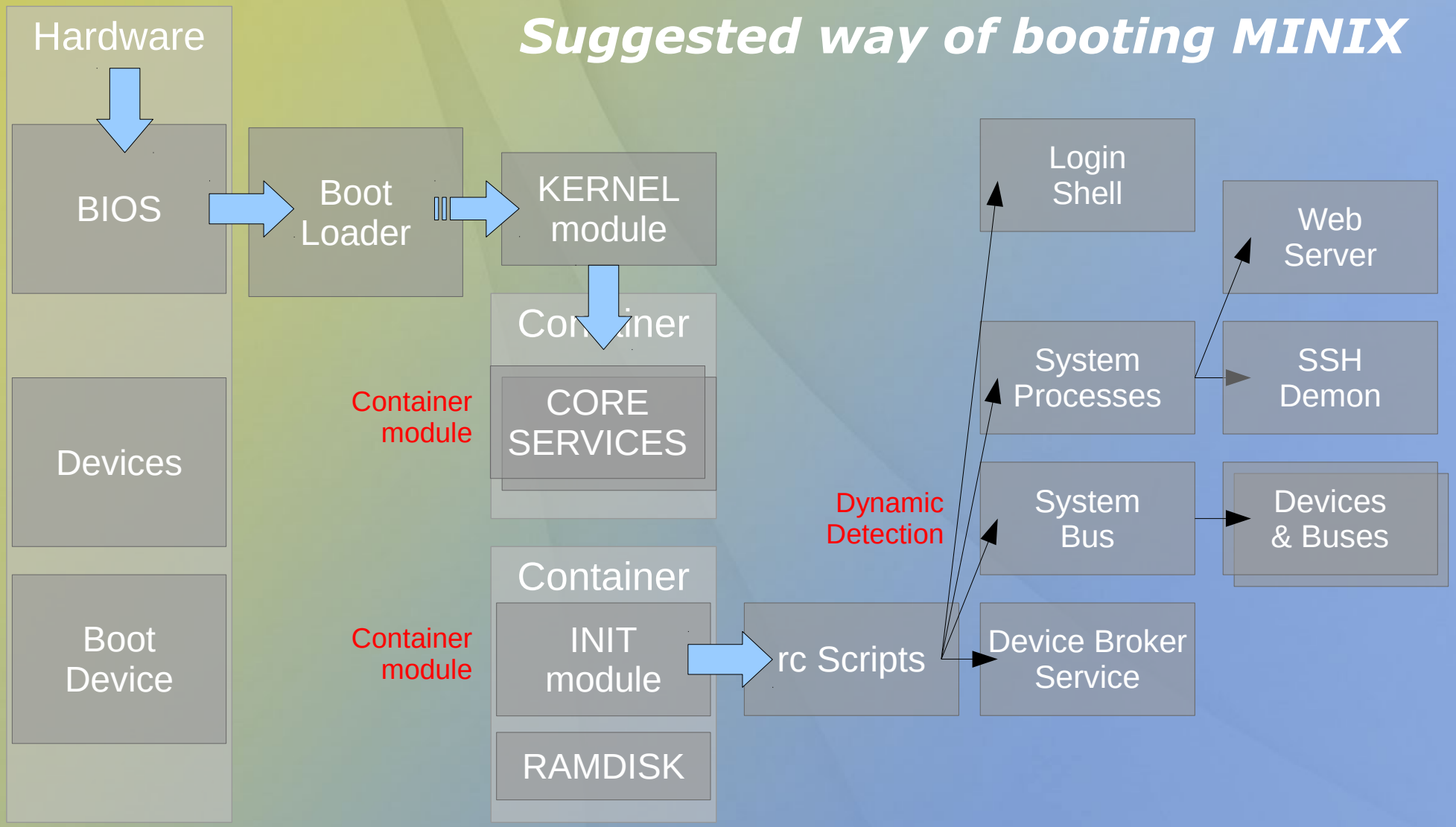
- Mass storage and file system are present after INIT
- Requirement for core services, buses and drivers before

Modularized booting?

Ideas ***Suggested changes***

- Any boot loader loads MINIX, core services, initial ramdisk
  - **Current status:** Multiboot v1 compliant boot loader
- Thin separation of MINIX kernel and boot headers
  - **Current status:** Multiboot data handled by kernel
  - **Idea:** Separate „boot data handling” and kernel
- Join core services into one module to simplify loading
  - **Current status:** Many separate modules, issue for e.g. iPXE/PXE
  - **Idea:** Use one module with offset table
- RAMdisk provides drivers to mount mass storage
  - **Current status:** PCI and pre-selected drivers
  - **Idea:** Dynamic bus/device loading incl. PCI and USB
  - **Idea:** Identical scripts in INIT ramdisk and later mounted disk

Dynamic booting?



Privilege Modes

- Monolithic system design (1)
- Modular system design (2)

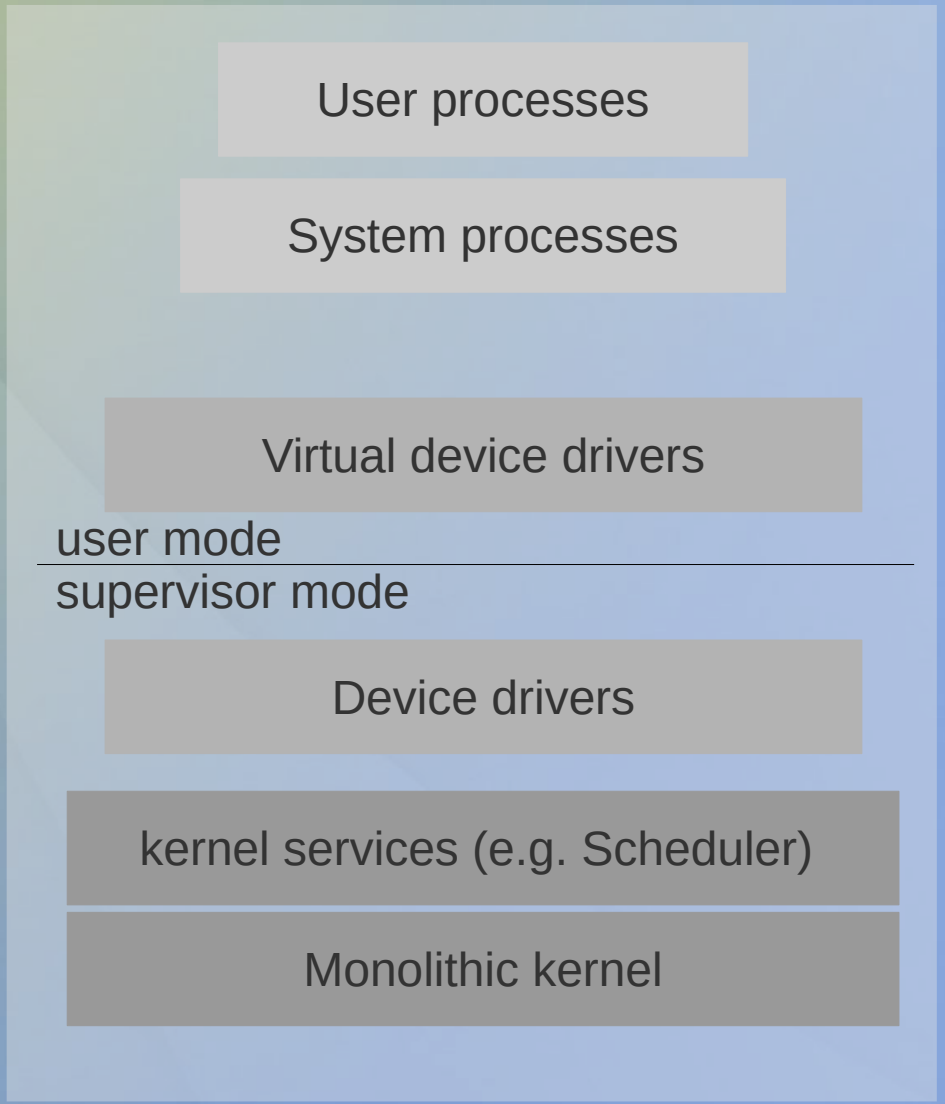
Privilege Modes (1)

Supervisor Mode

- Unrestricted hardware access
- Free physical memory access
- Any CPU instruction

User Mode

- No direct access to hardware
- No direct access to physical memory
- Virtual memory isolation
- System API restricts hardware access
- System API controls phyiscal memory access



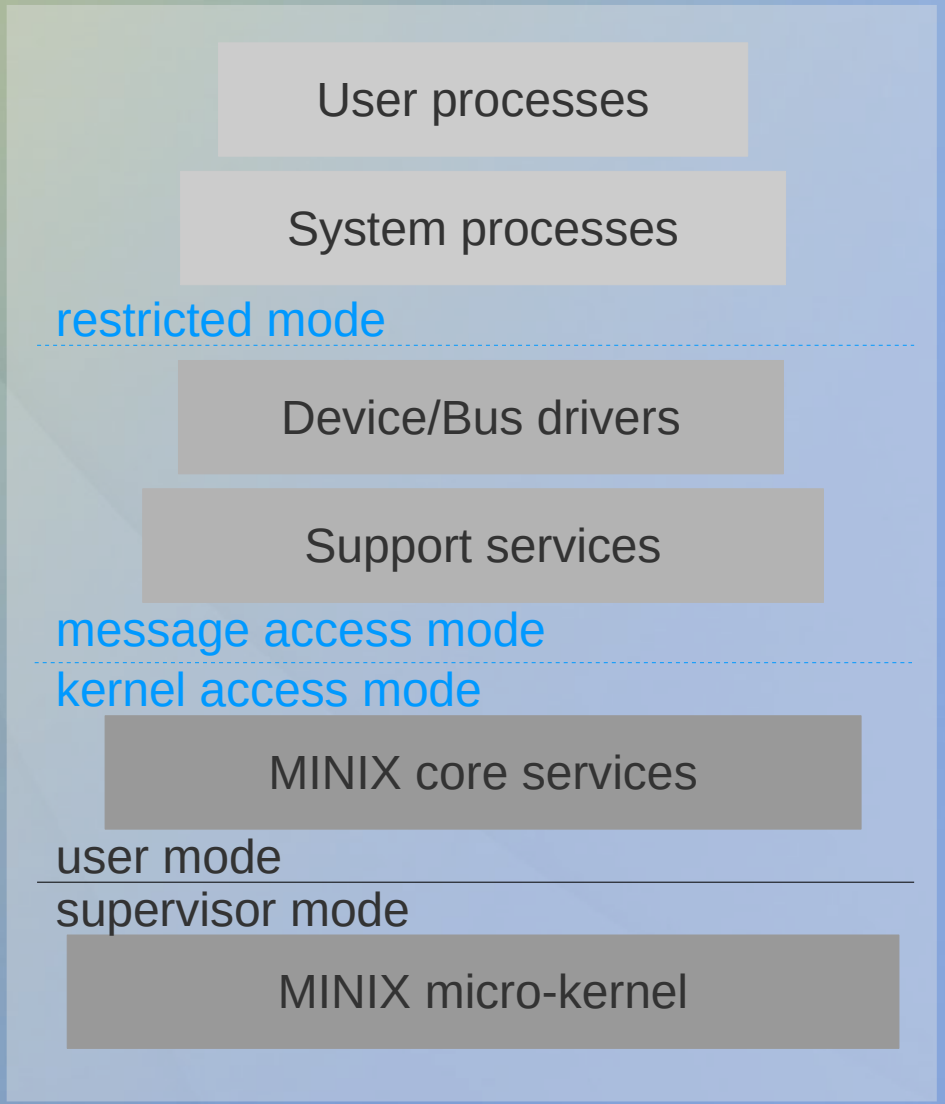
Privilege Modes (2)

Micro-kernel design

- Very small supervisor mode
- Additional protection layers

MINIX service design

- *Core services* with kernel communication
  - Virtual Memory Service
  - Scheduler and Process Mgr.
  - Reincarnation Service
- *Support services* with messages
  - System TTY console
  - Device Manager
- *Device drivers* with messages
  - PCI, SATA, etc.
- *User processes* with system API



Bus/Device Concept

- Generic view on devices (1) … (3)
- Separation of concerns (4)

Generic view on devices (1)

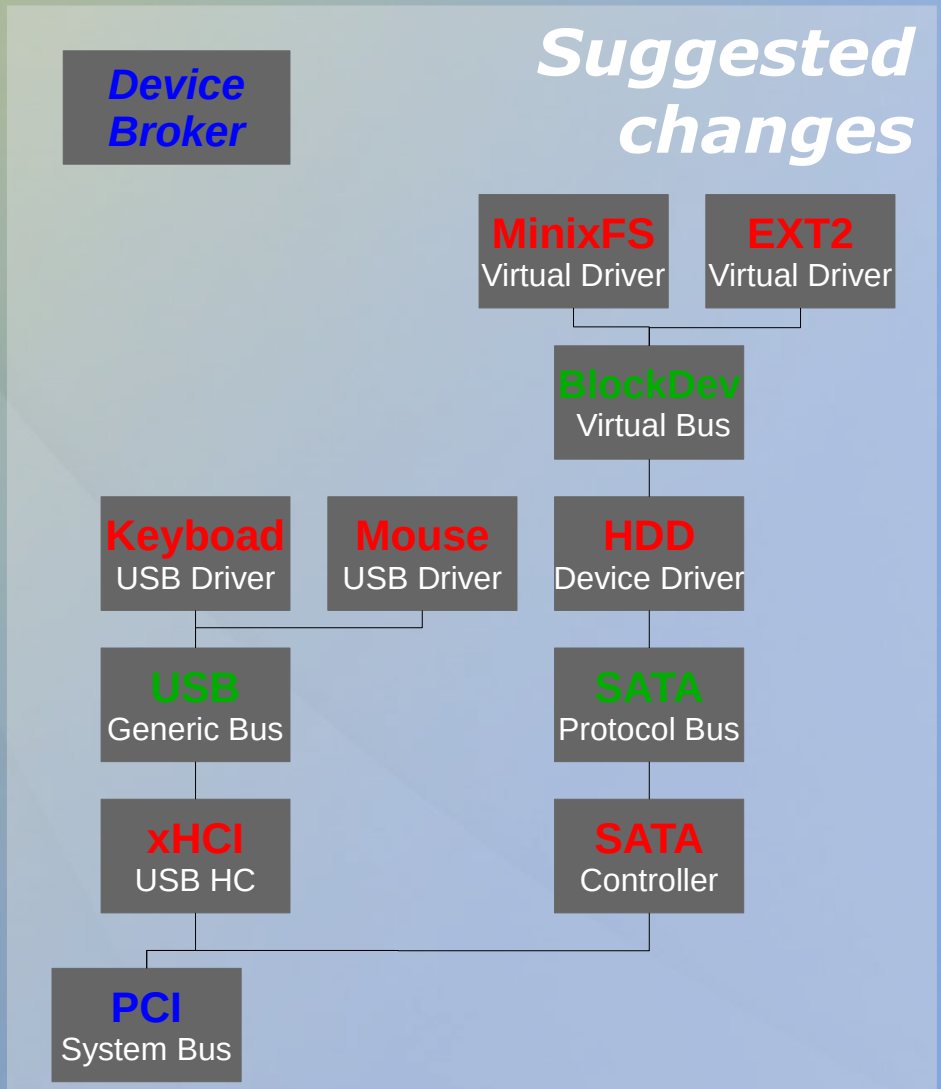
Ideas ***Suggested changes***

- Separation of components
  - Hardware handshake (classic driver)
  - Required communication protocol and functionality
- Keep layered devices separated
  - Bus as interface and middle layer
  - Bus provides enumeration of devices
  - Layered device communicate via the bus interface
- Device management „broker”
  - Devices and Buses register themselves
  - Usage is requested via the broker

Generic view on devices (2)

Example

- *Device Broker* as managing service
- Initial *System Bus*, e.g. PCI
- *Drivers* providing hardware handshake
- *Buses* providing protocol and functionality layer
- Every *Bus* or *Driver* instance registers at the *Device Broker*



Generic view on devices (3)

Benefits ***Suggested changes***

- New hardware relies upon existing interfaces
  - e.g. Keyboard/Mouse on USB share common code with system bus equivalent
  - e.g. any disk on USB or SATA or Network share same code via „block device bus”
- Smaller implementation risks on adding new devices
  - Existing and tested functionality code will be used
- Less effort on introducting new bus systems
  - Providing a shared interface allows plugin of existing devices

Separation of concerns

Expected modular (service oriented) design

- New drivers provide clear and common interfaces
- Bus systems provide common interfaces
- Interfaces are generic for device types
  - e.g. all block devices provide the same interface
- Device driver does not care for later use of I/O data
  - i.e. implements hardware handshake only
- Virtual driver does not access hardware directly
  - i.e. implements protocol and/or functionality

Thank you for listening.

Contact:

Bernd Onasch
dieCobol.de GmbH
Karlsruhe, Germany
bernd.onasch@diecobol.de

Additional Slides

- Glossary
- References

Glossary

KERNEL

- Exclusively the MINIX kernel module

SYSTEM CORE

- MINIX kernel module and core services

CORE SERVICE

- Mandatory service on system boot
- Service required to have the system running

SERVICE

- Privileged process providing a service to the userland
- Service process can be „reincarnated”

DEVICE

- Privileged process interfacing a hardware component

BUS

- Privileged process communicating with device(s) and providing a bus structure

References

Boot Process

http://wiki.minix3.org/doku.php?id=developersguide:frompowerontologinprompt

ftp://download.intel.com/design/archives/wfm/downloads/pxespec.pdf

http://ipxe.org/docs

Privilege Modes

https://en.wikipedia.org/wiki/Protection_ring

http://www.linfo.org/kernel_mode.html

Bus/Device Concept

http://git.minix3.org/index.cgi?p=minix.git;a=summary (Status from 2016-01-12)

http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/extensible-host-controler-interface-usb-xhci.pdf

etc.