Minix 3

going from educational showcase to day-to-day productive use

History of micro kernels

- Early approaches
  - IBM's hypervisor launching CP/CMS
- Microkernel designs
  - Mach 3 *(microkernel)*
    - □XNU *(hybrid kernel)* lauching the OS
    - ▯ GNU Hurd
  - MINIX *(microkernel)*
  - Plan 9 *(concepts)*
- Specific implementations
  - AmigaOS

Comparison monolithic / micro

Micro kernel

- Bootstrap mechanism
- Address space handling
- CPU allocation
- Interprocess Communication (IPC)

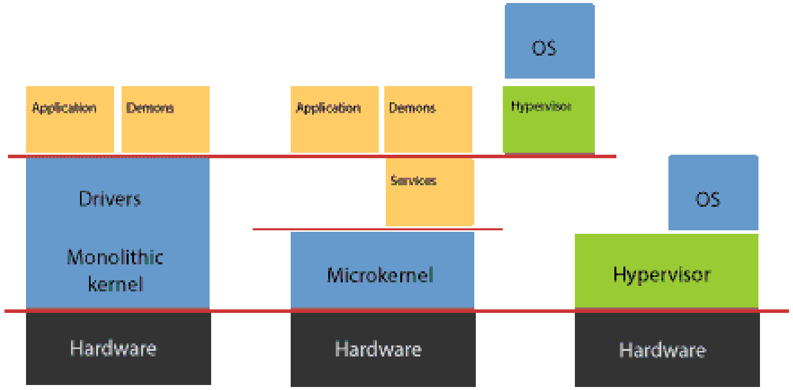
Monolithic kernel

- All from micro kernel plus
- Memory Manager
- Scheduler
- Device drivers
- Basic file system

Comparison monolithic / micro

Hypervisor

- Based on „any“ kernel plus
- Virtualisation layer for
  - any hardware component
  - any virtual device



Key principles

- Its all about abstraction!
    - Top down approach
    - Implicite hierarchy
    - Model
- Focusing following principles
    - Separation of Concerns
    - Components

Separation of Concerns

- Mentioned first by Dijkstra (1974)
- Makes the abstraction more cohesive
    - Weaker coupling between „modules“
    - Explicite (abstract) contract for using the module
- Commonly used in many engineering domains
    - Layer-based approaches, e.g. block device/file system
- Leverages abstraction
    - As a result you also get a hierarchy for free!
- Drill down process ends typically too early!

Components

- A Component encapsulates related functions
- Abstract interface and implementation
  - Top-down approach, interface first!
- Encapsulation
  - State is manipulated via interface (contract) only!
- Information hiding
  - No implementation details are exposed
- Own lifecycle
  - Starting/stopping/replacing is possible during runtime

Prerequisites/Paradigm

- Abstraction!
  - Top down approach
- Explicite interface first!
  - Part of top-down
- Component driven
  - Loose coupling, strong cohesion!

Current situation

- Very specific bare metal installation
  - Hard to install on up-to-date hardware
- Programmed against very specific hardware
  - Intel is widely spread, but not the standard

Installation

- Very specific bare metal requirements
  - X86: IDE/SATA CD/DVD-ROM
  - BB: sd card only
- Its not possible via
  - Plain PXE (ipxe make the trick with a fix)
  - USB
- Hard to install on SBC

HW support & implementation

- Programmed against very specific hardware
  - Vendor specific details are mandatory
- Concrete examples
  - kbd driver paniced, if no kbd HW found
  - PCI driver expects a device at bus 0, device 0

How to make it better

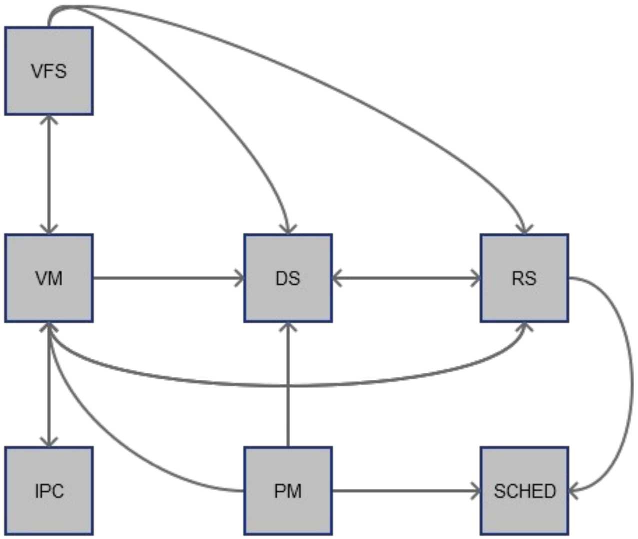
- Kernel
- Services
- Damons

Kernel

- Separation of Concerns
  - Memory model
  - Instruction set
- Results
  - Reuse the memory model code
  - Make the instruction set relevant code smaller

Services

- Circular dependencies
  - VM is circular with IPC, RS and VFS
  - RS is circular with DS and VM
- Most startup services are interdepend with each other



Services

- Resolve cyclic dependencies
- Implement drop-in replacement scheduler
- Allow multiple instances of services
- Separate scheduling for „real-time“ and „batch processing“

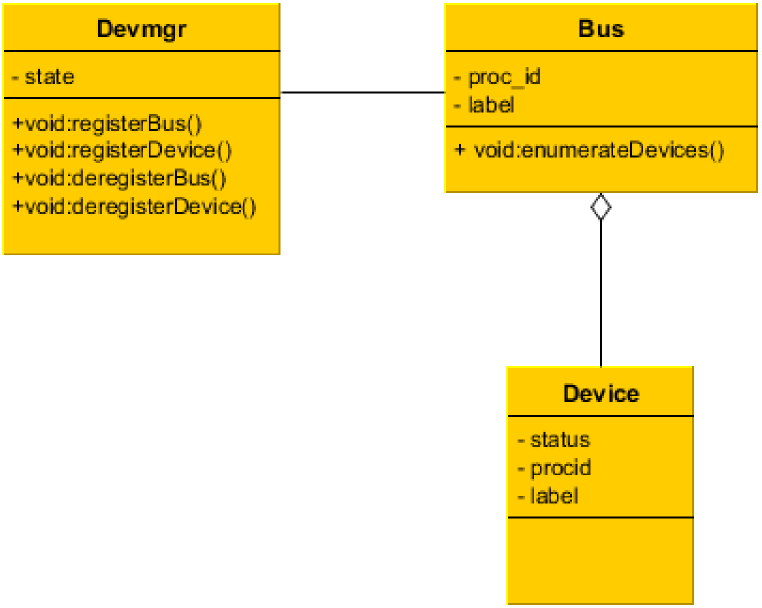
Demons

- USB for minix arm
  - Quite monolithic
  - One process contains HC and USBD
- Drill down the SoC
  - One service per host controller
  - Host controller registers at USBD
  - Necessity for x86 based USB ([E|O|U|X]HCI

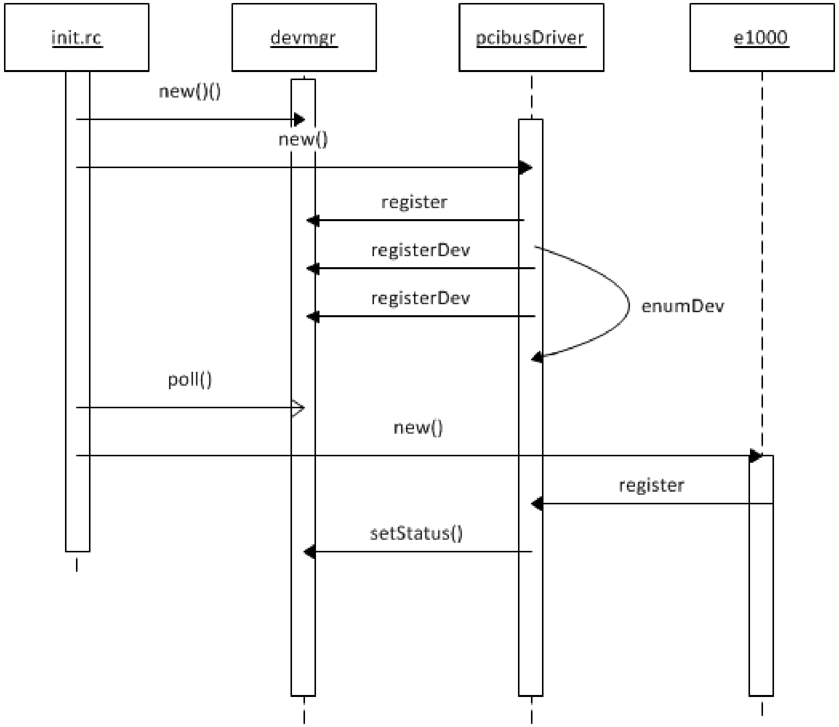
Concrete example - DevMgr

- Contains busses
- Devices belong to busses
- Hot plug

Abstract devmgr model



Devmgr sequence diagram



Functionality of devmgr

- Single point of authority
  - Tree structure for bus and devices
  - Status (driver loaded/shared/exclusive)
- Simplifying dependencies
  - Virtual devices will work, too! (file system)

Conclusion

- Need to accept the paradigm
- The right abstraction makes you powerful