Introduction to the Linux Kernel

Praktikum Kernel Programming

University of Hamburg
Scientific Computing

Winter semester 2015/2016

Konstantinos Chasapis
Konstantinos.chasapis@informatik.uni-hamburg.de

Outline

- Introduction (story, licence, versioning)
- Main parts
- Loadable Kernel Modules
- System Calls
- Security

Introduction

- Developed by Linus Torvalds (1991)
  - Just for Fun: The Story of an Accidental Revolutionary by Linus Torvalds
- Based on Unix
- 1st version supported Intel 80386
- Currently various platforms are supported
- Implemented in GNU C
- Several Distributions (distro)
  - RedHat, CentOS, Ubuntu, SUSE, Debian, Arch
  - Different package system, configuration etc.
  - Apply different patches

Introduction (cont.)

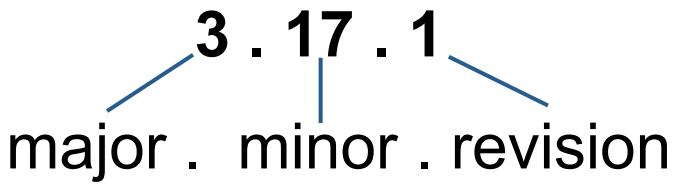
- X-Server is not implemented within the Kernel
- Everything run in “Kernel mode”
  - Privileged access to hardware
- Monolithic but boasts modular design
  - Kernel preemption (under certain conditions)
    - The scheduler is permitted to forcibly perform a context switch
  - Supports kernel threads
  - Dynamic load and unload binaries (kernel modules)
  - Reentrant, several processes can be in kernel mode simultaneously

Introduction (cont.)

- License Terms
  - is licensed under the Version 2 of the GNU General Public License (GPL)
  - Allows anybody to redistribute and even sell a product covered by GPL as long as the recipient has access to the source and is able to exercise the same rights
  - Any software derived by a product covered by GPL must be released under the GPL
- Democratize, everyone can contribute
  - If you want your code to go into the mainline or you have modified the kernel then you have to use GPL-compatible license

Introduction (cont.)

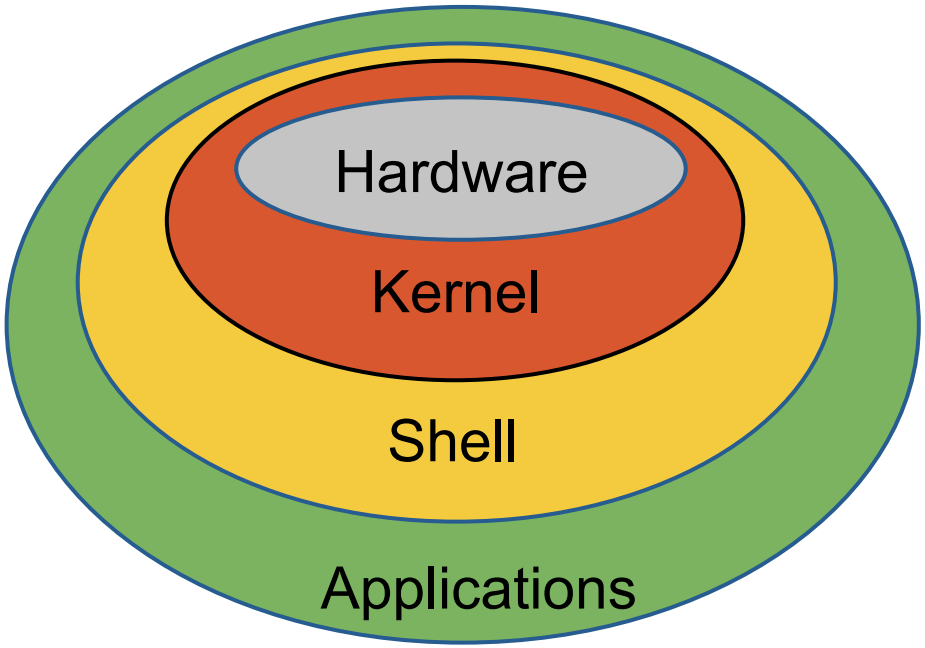
- Use of binary Blobs (Modules, firmware)
  - The source is not given
  - May contain part of the driver from another file system
  - If the code has been ported from another operating system is legal
  - If a company wants to keep the source private
  - Using such software is discourage
- Versioning
  - $uname –a



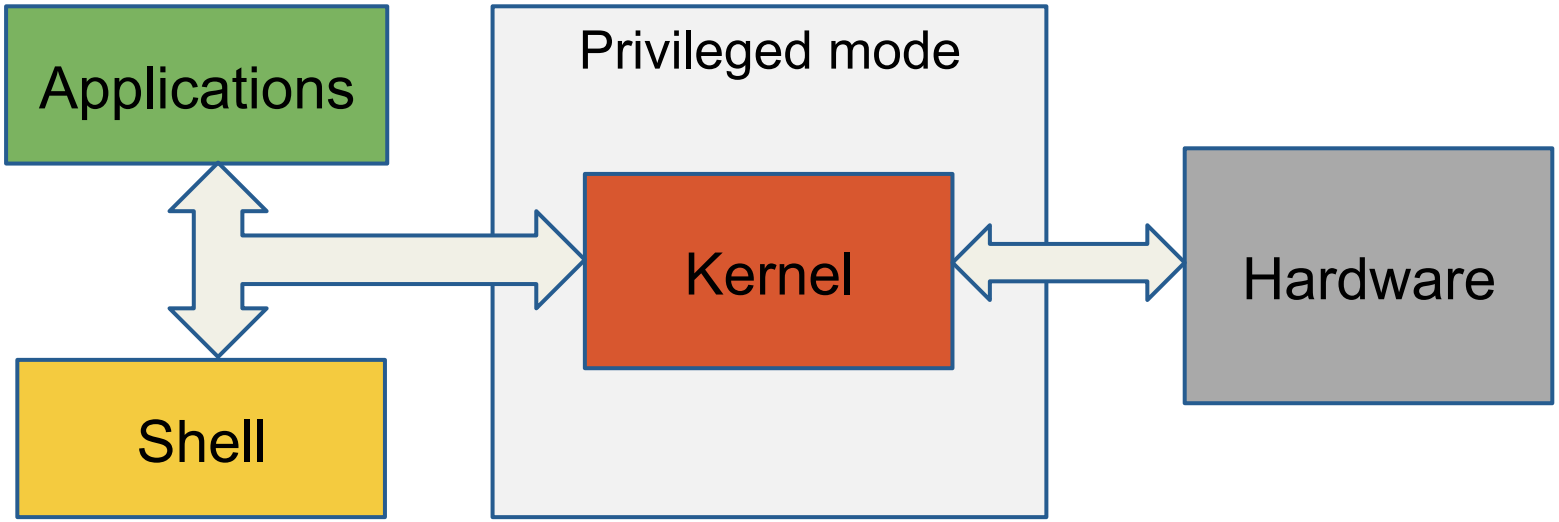
Outline

- Introduction (story, licence, versioning)
- Main parts
- Loadable Kernel Modules
- System Calls
- Security

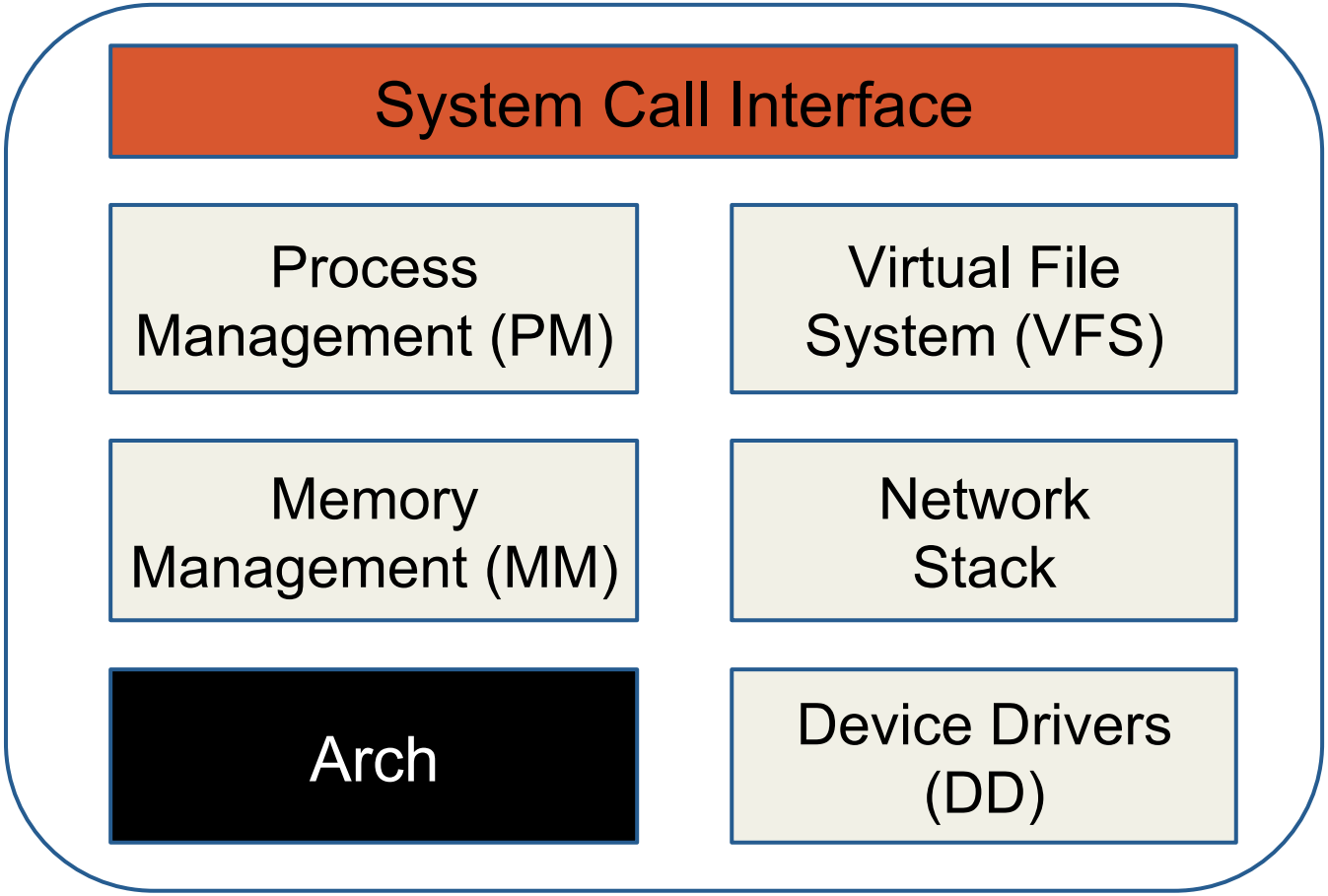
Linux system overview



Request flow



Main parts

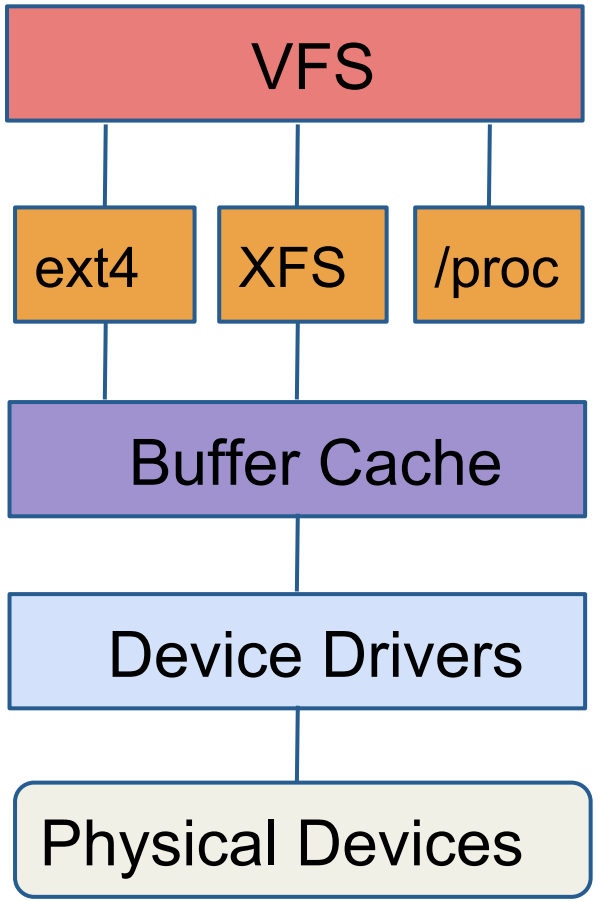

System Call Interface
Process Management (PM)
Virtual File System (VFS)
Memory Management (MM)
Network Stack
Arch
Device Drivers (DD)

Main parts (cont.)

- System call interface (SCI)
  - A thin layer that provides a method to interact from user space to kernel space
- Process Management (PM)
  - Create, destroy processes
  - Communication between different processes (kernel threads)
  - CPU scheduling
- Memory Management (MM)
  - Physical to virtual memory management
  - Memory allocation
  - Swapping, from memory to hard disk

Main parts -- I/O Path

- Virtual File System (VFS)
  - Eports the common file interface
  - Abstract file system functionality from implementation
- File Systems
  - Implementation of FS functionality
- Buffer Cache
  - A set of functions to manipulate main memory designed for FS
- Device Driver
- Physical Device
  - Where data live



Main parts (cont.)

- Network Stack
  - Implement the network protocols
  - Deliver packets across programs and network interfaces
- Device Drivers (DD)
  - Interact with the hardware
  - Extract an abstraction of the device functionalities
- Arch
  - Architecture dependent code

Outline

- Introduction (story, licence, versioning)
- Main parts
- Loadable Kernel Modules
- System Calls
- Security

LKMs

- LKMs (Loadable Kernel Modules)
- Pre-compiled binary pieces
- Each piece is called “module”
- Can be loaded at runtime
- Extend the functionality of the system
- Enforce modularity
  - Easy to develop, debug and maintain
  - No need to rebuild the kernel
- Can save memory (load only the necessary)

What are LKMs used for

- Everything that is not required in the core
- 6 main categories
  - Device drivers
  - File system drivers
    - Implementation of a specific file system
  - System calls
  - Network stack
    - Interprets a network protocol
  - TTY line disciplines
  - Executable interpreters for the supported formats

Character Device Driver

- Read or Write a byte at a time
- Accessed by a stream of bytes
- Usually permit only sequential access
- Implement: open, close, read, write
- Similar to regular files
  - /dev/console
  - /dev/ttyS0

Block Device Driver

- Read or Write block-size multiples
- Permit random access
- Accessed in the /dev/
- File systems can be mount on top
- Handle I/O operations
- Differ with the char module in the way the manage data inside the kernel
- Different interface to the kernel than char modules

Network Drivers

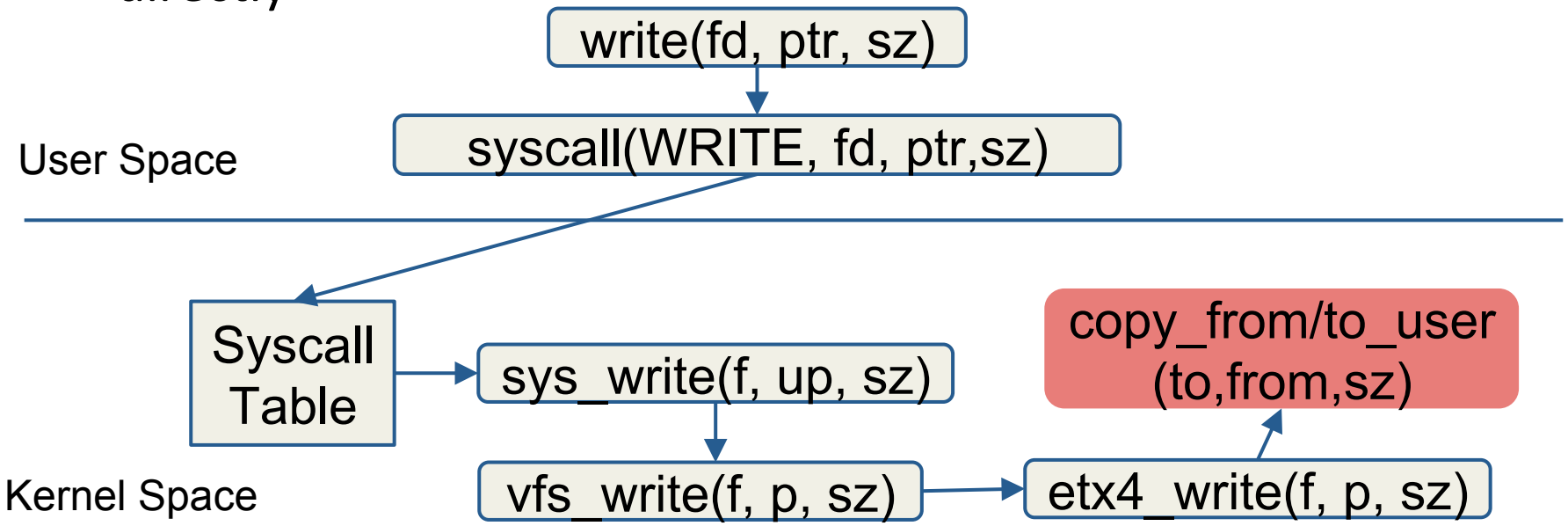
- Handle any network transaction made
- Transfer packets of data
- Independent of a specific protocol
- Reception and Transmission instead of Read/Write
- Usually the interface is a hardware device but it can also be software like the loopback
  - loopback is used to communicate with the servers that run in the same node, debugging etc.
- They are not mapped to the file system; they are identified by a name

Outline

- Introduction (story, licence, versioning)
- Main parts
- Loadable Kernel Modules
- System Calls
- Security

System calls

- A syscall causes a programmed exception (trap) on the CPU
  - syscall(number, arguments)
- Within the kernel you cannot access user space buffers directly



Outline

- Introduction (story, licence, versioning)
- Main parts
- Loadable Kernel Modules
- System Calls
- Security

Security considerations

- Security check is enforced by the kernel
- If the Kernel has „holes“ ⇨ System has holes
- Avoid introducing typical programming bugs
  - Module parameters
  - Buffer overrun
  - Memory corruption
- Zero or initialize memory given to user
- Run precompiled kernels found in your distro
- In official distros only the superuser can load and unload modules

Kernel programming is vital for as long as new hardware is being designed and produced or old-obsolete hardware is maintained.