BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

3

OOP – Linux

Boot-Vorgang und Systeminitialisierung

mit Skriptmaterial von Dr.-Ing. M. Feldmann

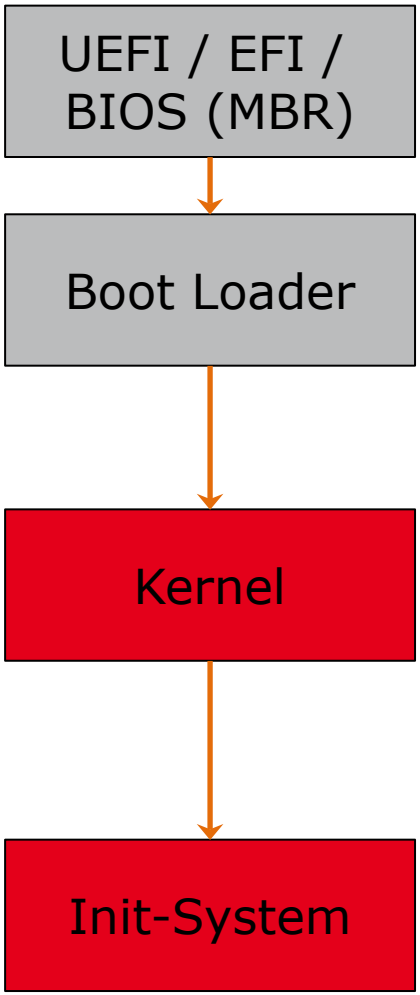
Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-dresden.de

Gliederung der Vorlesung

Inhalte

- Übersicht
- UEFI
- Bootloader/GRUB
- Initramfs
- Init-System
- Init-V/Systemd

Übersicht zum Boot-Vorgang



- (U)EFI = (Unified) Extensible Firmware Interface
- startet Boot Loader von einem Medium (z.B. HDD)

- verweist auf verfügbare Kernel
- initialisiert Kernel
- Übergabe von Parametern an Kernel möglich

- initialisiert Subsysteme wie Speicherverwaltung, Task-Scheduler
- Startet das Init-System
  (Verweis auf Pfad ist in den Kernel-Code integriert)

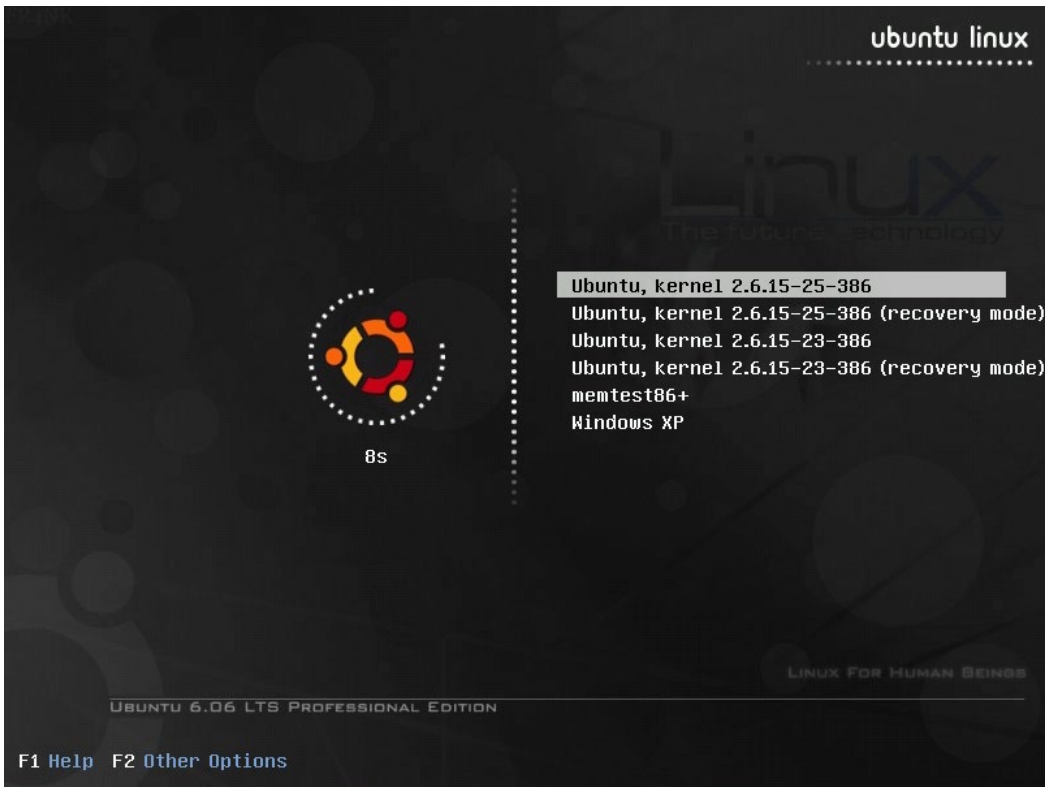
- Startet alle erforderlichen Nutzerprozesse im sogenannten „User-Space“ gemäß der Konfigurationsvorgabe
- Verschiedene Alternativen verfügbar

UEFI

- UEFI bildet den Ersatz für das BIOS (Basic Input/Output System)
- umfangreicher Funktionsumfang, u.a.
  - integrierte Kommandozeile
  - Zugriff auf Partitionstabellen und Dateisysteme
  - PXE (Preboot eXecution Environment)
  - Netzwerkfunktionalität
  - Secure Boot
- Information für Boot Loader liegt in EFI System Partition (ESP)
  - Linux: z.B. `/boot/efi/EFI/GRUB/`
  - Windows: Partition 0 mit eigenem Einhängepunkt (`mountvol /s`)
- Standard-C-Funktionen können in UEFI-Applikationen genutzt werden
- Unterstützung für UEFI kann für Linux aktiviert werden mittels Kernel-Option `CONFIG_EFI_PARTITION`

Boot Loader

- Gängigster Linux Boot Loader ist der Grand Unified Bootloader (GRUB)
  - in unterschiedlichen, parallel entwickelten Versionen (Legacy und 2)
  - https://www.gnu.org/software/grub/



https://commons.wikimedia.org/wiki/File:GRUB.png

- Alternativen u.a. LILO, Syslinux, efilinux

GRUB

- Konfiguration für Bootmenü-Einträge und -Optionen für GRUB befindet sich in Konfigurationsdatei unter `/etc/grub`
- Generierung der Konfigurationsdatei durch `grub-mkconfig` (Ausgabe erfolgt aber auf STDOUT)
- Werkzeug ruft die in `/etc/grub.d/` enthaltenen Skripte auf
- Änderung der Boot-Parameter in GRUB-Laufzeit möglich (Taste „e“)
- sollte das System nicht in der Lage sein zu booten, ermöglicht GRUB meist eine manuelle Korrektur über eine dedizierte Konsole (Taste „c“)
- GRUB-Konsole unterstützt diverse Funktionen, u.a.:
  - `ls` Auflisten der bekannten Geräte
  - `echo $root` Zeigt aktuelle Wurzel des Dateisystems an
  - `ls ($root)/boot` Anzeige der Inhalte des boot-Verzeichnisses
  - `set` Anzeige der GRUB-Parameter (Setzen von Parametern ebenfalls möglich)

GRUB

- Installation des GRUB im MBR (für BIOS)

```
user@linux$ sudo grub-install $device # z.B. device=/dev/sda
```

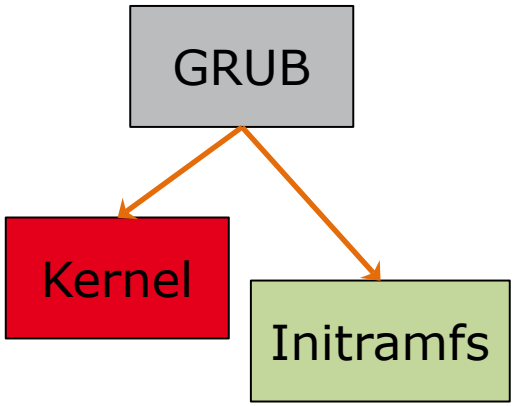
- Installation des GRUB in der (U)EFI-Partition

```
user@linux$ sudo grub-install
            --efi-directory=$dir
            --bootloader-id=$name
```

Zum Kommando `sudo`:

- ermöglicht Ausführung von Superuser Befehlen aus dem Nutzerkontext
- mehr dazu im Kapitel „Ausgewählte Themen der Systemadministration“

INITRAMFS




```
menuentry 'Ubuntu‘ {
    # Vor dem Laden des Linux-Kernels
    # können Module für RAID, GPT, etc.
    # geladen werden...
    linux /vmlinuz-3.16.0-0.bpo.4-amd64
    initrd /initrd.img-3.16.0-0.bpo.4-amd64
}
```

- Initial RAM File System als Nachfolger der Initial RAM Disk
- unterstützt Modularisierung des Kernels
- dient hauptsächlich dazu, Kernelmodule zu laden, die für die Systeminitialisierung erforderlich sind (z.B. Dateisystemtreiber)
- Distributionsabhängig konfiguriert, z.B. via `/etc/initramfs-tools`
- typischerweise in Datei `/boot/initrd.img.-KERNELVERSION`
  → Inhalte entpackbar via

```
user@linux$ zcat /boot/initrd.img-$(uname -r) | cpio -idmv
```

Start des Init-Systems

- Kernel-Initialisierung beginnt nach einigen Vorbereitungen mit dem Aufruf der Funktion `start_kernel()` in `init/main.c` (siehe Git-Repository auf kernel.org)
- nach Initialisierung aller erforderlichen Subsysteme ruft diese die Funktion `rest_init()` auf, was wiederum `kernel_init()` aufruft:

```
static int __ref kernel_init(void *unused) {
    # ...
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh")) return 0;
    panic("No working init found."
          "Try passing init= option to kernel."
          "See Linux Documentation/admin-guide/init.rst"
          "for guidance.");
}
```

Init-System

- Kernel startet das Programm `/sbin/init`
- Init-System wird innerhalb des ersten Prozesses des Systems mit Prozessidentifikationsnummer (PID) 1 ausgeführt
- nachfolgender Ablauf abhängig von konkretem Init-System (3 Varianten verbreitet)
  - Init-V-System
  - Upstart
    - ereignisbasiertes System
    - alle Dateien in `/etc/init` werden gelesen und ausgeführt
  - Systemd
    - neben Init-Funktionalität besitzt es zahlreiche weitere Werkzeuge für verbreitete Systemaufgaben
    - in vielen Distributionen zum Default-System geworden

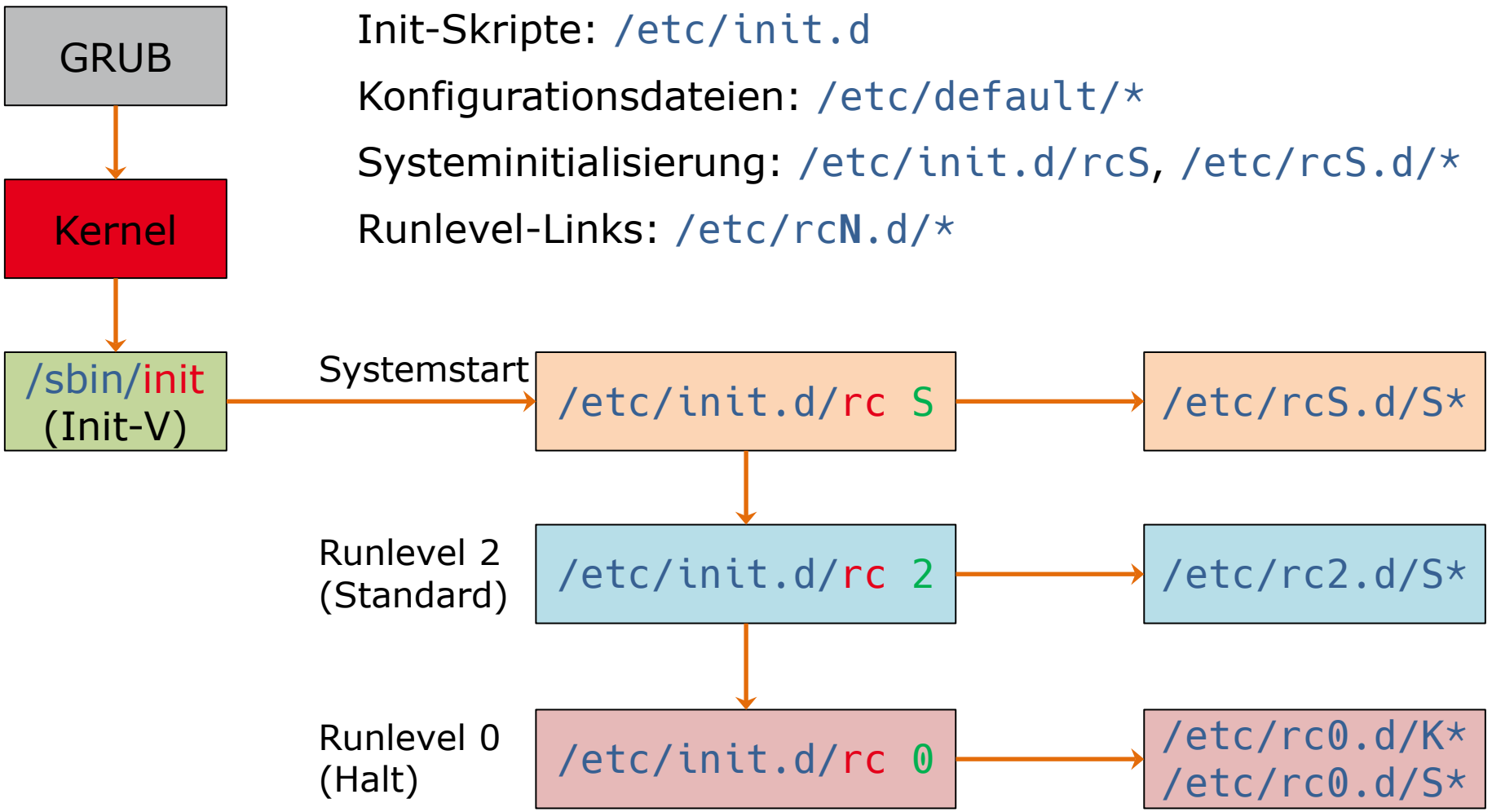
Identifikation des Init-Systems

- Zur Identifikation, welches Init-System läuft, kann geprüft werden, auf was der jeweilige Pfad zu `init` verweist:

```
user@linux$ ls -al /sbin/init
lrwxrwxrwx 1 root root 20 Jul 19 01:56 /sbin/init -> ../lib/systemd/systemd
```

- Die Manpage für den Befehl `init` ist typischerweise für das verwendete Init-System
- Zusätzlich:
  - Upstart hält in `/etc/init` bei der Initialisierung zu ladende Konfigurationsdateien (`.conf`-Dateien)
  - Systemd verwendet dafür `/usr/lib/systemd` und `/etc/systemd`
  - System-V verwendet dafür `/etc/inittab`

Schema für Init-V



SYSTEMD

- Standard-Init-System in den meisten Distributionen
- Systemd generalisiert die Entitäten, die es kontrolliert zu Units
- Unterscheidung mehrerer Typen von Units, die typisch für Unix-Umgebungen sind
  - Service Units: kontrollieren Daemonen und Prozesse
  - Timer units: für das zeitbasierte Ausführen
  - Mount Units: für das Einhängen von Geräten in das Dateisystem
  - Target Units: kontrolliert weitere Units (typischerweise durch Gruppierung)
  - für weitere siehe Manpage von Systemd
- Units liegen unter `/usr/lib/systemd/*`
- initial wird `default.target` angesteuert, das weitere Abhängigkeiten besitzt und damit das Gesamtsystem initialisiert
- Abhängigkeitstypen/Beziehungen zwischen Units: requires, wants, requisite, conflicts und andere

SYSTEMD – EINFACHES BEISPIEL

- Ablage der folgenden Dateien in: /etc/systemd/system/

example1.target

```
[Unit]
Description=This is a first example...
```

example2.target

```
[Unit]
Description=This is a second example…
Wants=example1.target
```

- Units können gestartet werden mittels `start` (inverse Operation: `stop`):

```
user@linux$ sudo systemctl start example2.target
```

- Abfrage des Status der Units:

```
user@linux$ sudo systemctl status example1.target example2.target
```

- In einer separaten Sektion `[Install]` können ebenfalls Abhängigkeiten ausdefiniert werden, hierzu muss die Unit dann erst aktiviert werden (mittels `systemctl enable $unit`)

Aufgaben

1. Verschaffen Sie sich einen Überblick über das Verzeichnis `/proc/1`. Welche Informationen finden sich in diesem Verzeichnis?
2. Generieren Sie eine GRUB-Konfiguration, die in der Datei `$HOME/grub.cfg` gespeichert wird.
3. Erstellen Sie ein Systemd-Unit-File, um ein einfaches Shell-Skript zu starten, das in regelmäßigen Abständen den String „Hello, I am alive.“ in eine bestimmte Textdatei (durch Sie festzulegen) schreibt.