Konzepte von Betriebssystemkomponenten

Systemstart und Programmausführung

Seminarvortrag

15.12.2003, Michael Moese

Übersicht

2. Systemstart
3. Programmausführung

TEIL 1: Systemstart

1.1 Das BIOS

1.2 Der Bootloader

1.3 Die setup()-Funktion

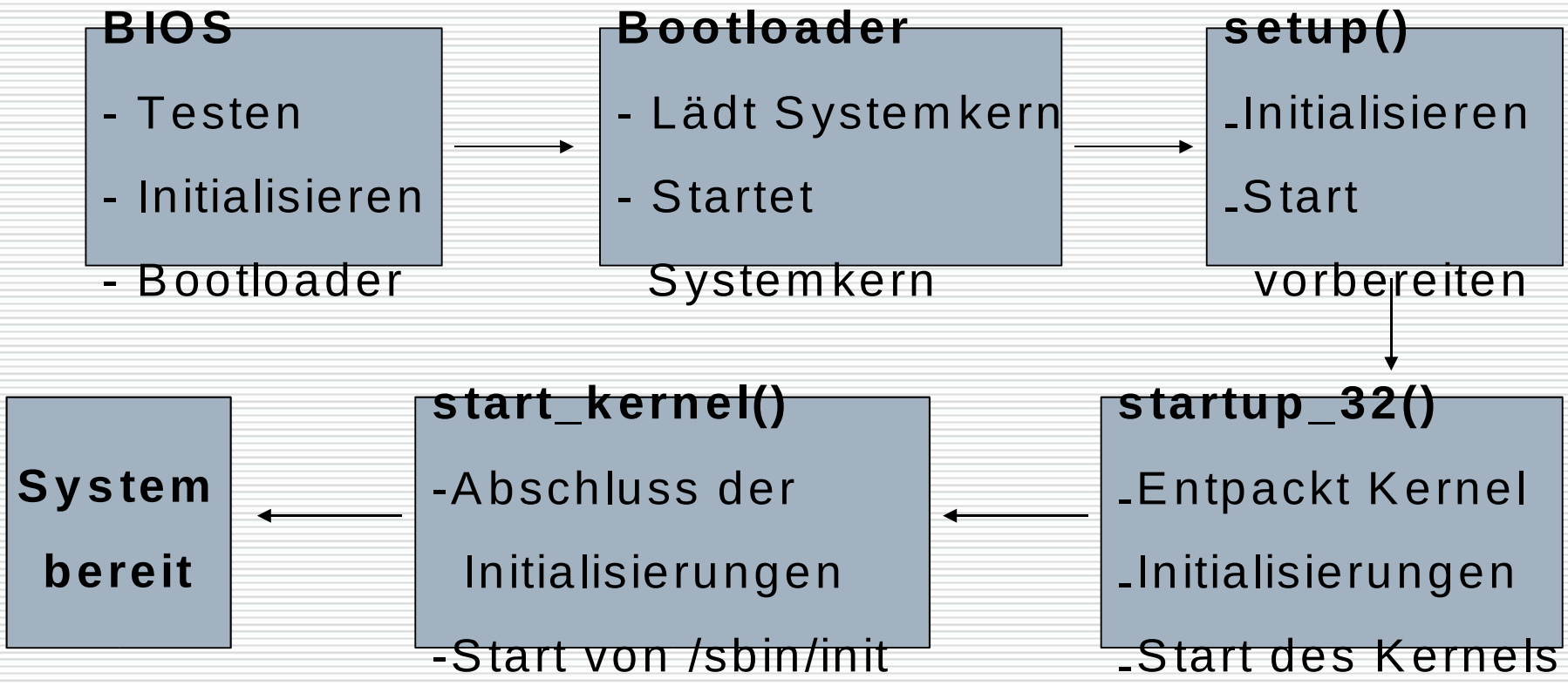
1.4 Die startup_32()-Funktionen

1.5 Die start_kernel()-Funktion

1. Systemstart

- Nach Einschalten: inkonsistenter, zufälliger Zustand des Systems
- Hardware muss getestet und initialisiert werden
- Start eines Bootloaders
- Bootloader lädt Systemkern
- Systemkern initialisiert das System fertig und startet sich.

1. Systemstart: Ablaufdiagramm



1.1 Das BIOS

Power On Self Test:

Test der Hardware (Grafikkarte, Speicher, Tastatur, Controller, uvm.)

Initialisierung der Hardware

Konfiguration der Komponenten, so dass sie sich IRQ‘ s und I/O-Ports teilen können bzw. keine Konflikte auftreten hierbei.

1.1 Das BIOS (forts.)

Suche des Betriebssystems

Reihenfolge kann der Benutzer bestimmen

Start eines Systems

wird vom ersten Datenträger geladen auf dem ein System gefunden wurde

1.2 Der Bootloader

Wird vom BIOS geladen

Lädt eigentliches Betriebssystem

Start von Diskette:

Bootloader lag im ersten Sektor, wurde geladen, lädt restliche Sektoren die Kernel enthalten

1.2 Der Bootloader (forts.)

Start von Festplatte:

Im ersten Sektor (MBR) liegt ein eigener kleiner Bootloader der das System von einer der Partitionen startet.

Linux verwendet anstatt dessen LILO oder GRUB.

Start via Netzwerk:

PXE, TFTP: Server im Netzwerk enthält Kernel und alles was zum booten notwendig ist. Bootloader (oder Boot-ROM) booten das System via LAN.

1.3 Die setup()-Funktion

- Initialisierung der Hardware
  (Linux verlässt sich nicht auf das BIOS)
- Anlegen einer provisorischen GDT und einer provisorischen IDT
- Umschalten der CPU in den Protected Mode, Paging noch deaktiviert.

1.4 Die startup_32() Funktionen

- Initialisierung der Segmentierungsregister
- Einrichten eines provisorischen Stacks
- Ungenutzten Kernel-Speicher mit Nullen initialisieren
- Entpacken des Kernel-Images

1.4 startup_32() (forts.)

- Vorbereitung des Ausführungskontextes für Prozess 0
- IDT mit Null-Vektoren füllen
- BIOS-Daten in erste Speicherseite schreiben
- Sprung zu start_kernel()

1.5 Die start_kernel()-Funktion

- Finale Initialisierung von Page Tables, Page Descriptors, IDT und Slab Allocator.
- Setzen von Zeit und Datum
- Ausführung von /sbin/init

TEIL 2: Programmausführung

2.1 Überblick

2.2 Ausführung eines Programms

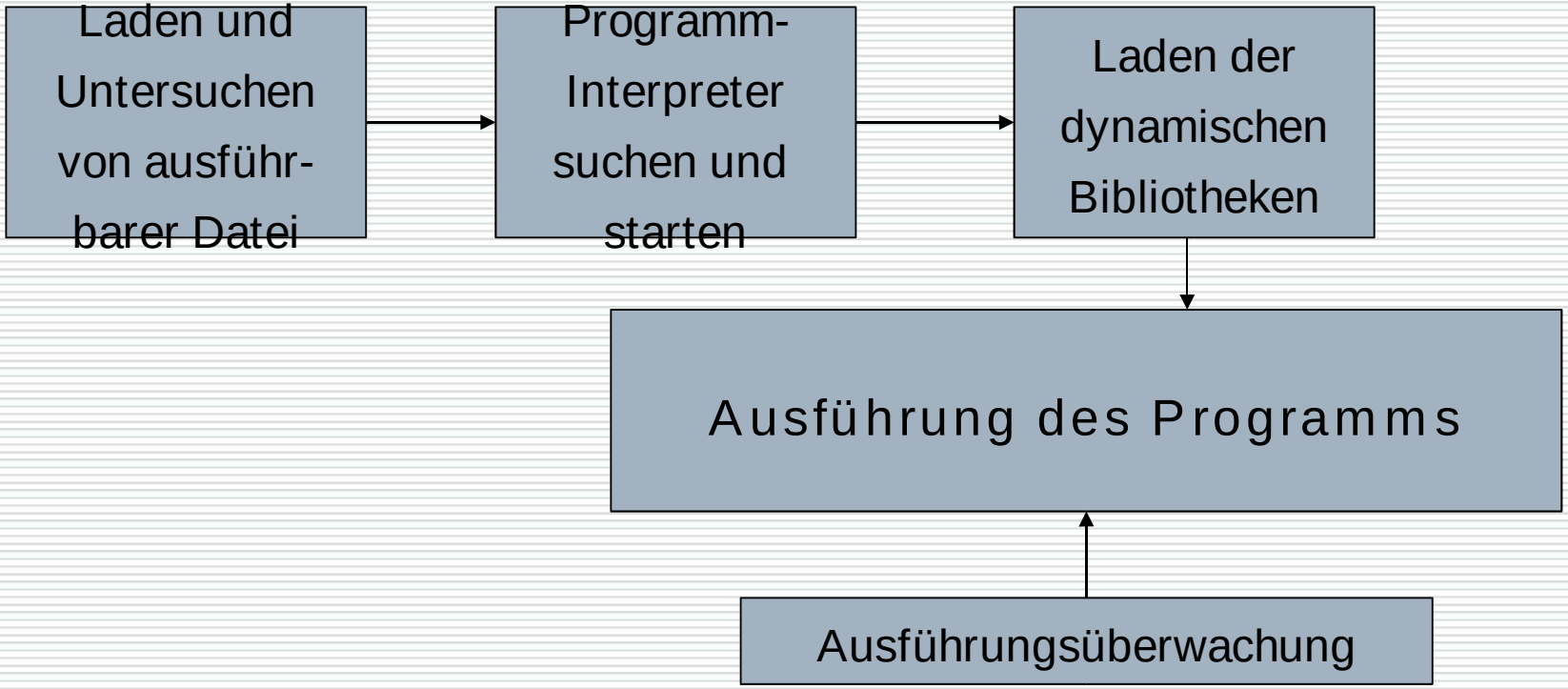
2.3 Formate ausführbarer Dateien

2.4 Programmsegmente

2.5 Bibliotheken

2.6 Ausführungsüberwachung

2.1 Überblick



2.1 Überblick (forts.)

- Programm: Anleitung, Rezept, um einen neuen Prozess anzulegen
- Verschiedene Formate für ausführbare Dateien
- Shared Libraries
- Ausführungsüberwachung

2.2 Ausführung eines Programmes

- Programmstart über eine der exec()-Funktionen
- Wenn ausführbar: Laden der ersten 128 Byte um Format zu identifizieren (Magic Number)
- Suche und Start des entsprechenden Programminterpreters

2.2 Ausführung eines Programmes

- Programminterpreter wird, ähnlich einem „ normalen“ Programm gestartet
- Programminterpreter analysiert Abhängigkeiten von shared Libraries
- Speicherbereiche werden angelegt
- Einblenden von Bibliotheken
- Sprung an Einstiegspunkt

2.3 Formate ausführbarer Dateien

Verschiedene Formate: ELF, a.out, COFF, EXE, uvm.

Enthalten Methoden zum Laden der Datei und Aufsetzen des Ausführungskontextes, für dynamisch gelinkte Bibliotheken und zum Schreiben des Ausführungskontextes in eine Datei (Core Dump)

2.3 Formate ausführbarer Dateien

Formate können unter Linux zur Laufzeit registriert werden.

POSIX-konforme Dateien kann Linux nativ ausführen

Andere (z.B. Windows-EXE) benötigen einen eigenen Interpreter.

2.4 Programmsegmente

- Textsegment
  enthält ausführbaren Code
- Segment für initialisierte Daten
  statische Variablen, initialisierte globale Variablen
- Segment für uninitialisierte Daten
- Stack-Segment
  (lokale Variablen, Rücksprungaddr., Aufruf-Parameter,..)

2.5 Bibliotheken

Compiler: erstellt Objekt-Datei aus einer Quellcode-Datei, erst Linker macht diese ausführbar

Statisch gelinkte Bibliotheken:
Bibliotheks-Funktionen sind komplett im Objektcode eines Programms enthalten

2.5 Bibliotheken (forts.)

Dynamisch gelinkte Bibliotheken (shared Libraries):

- Programme enthalten nicht die Funktionen, sondern lediglich Referenzen.
- Programminterpreter lädt Bibliotheken und blendet sie in den Adressraum des Prozesses (File Memory Mapping)

2.5 Bibliotheken (forts.)

Statisch gelinkte Programme:
Größere Objektdateien

Dynamisch gelinkte Programme:
längere Startzeiten
Womöglich Kompatibilitäts-Probleme

26 Ausführungsüberwachung

- Erlaubt es Programmen die Ausführung anderer Programme zu überwachen (Debugger)
- Überwachung eines Programms auf diverse Ereignisse
- Einzelschritt-Ausführung

Zusammenfassung

- Linearer Boot-Vorgang am Beispiel von Linux
- Komplexität des Vorgangs des Programmstartes