

Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. rer. nat. Peter Gerwinski

19. Mai 2025

Treiberentwicklung, Echtzeit- und Betriebssysteme

1 Einführung

2 Unix

3 Treiberentwicklung

3.0 Massenspeicher und Gerätedateien

3.1 Mikrocontroller

3.2 Betriebssysteme ohne Speicherschutz

3.3 Linux-Kernel-Module

3.4 Die Systembibliothek (libc)

3.5 Der Betriebssystemkern (Kernel)

4 Speicherverwaltung

...

3.3 Linux-Kernel-Module

- **#include** <linux/module.h>
#include <linux/kernel.h>
- Zwei „Hauptprogramme“: `init_module()`, `cleanup_module()`
- `printk()` statt `printf()`
- Compilieren mit speziellem **Makefile**
- Angabe der Lizenz ist wichtig: **MODULE_LICENSE()**
- Kommunikation mit Anwendungen:
Geräte-Dateien (*major/minor number*)
- Kommunikation mit dem Betriebssystemkern:
Callback-Funktionen (virtuelle Methoden)
- Automatisches Anlegen von Gerätedateien:
Geräteklassen

3.4 Die Systembibliothek (libc)

Verbindung zwischen Anwendung und Betriebssystemkern (Kernel)

- Verbindung zur Anwendung: Funktionsaufruf
Indirekter Funktionsaufruf über eine Tabelle von Zeigern, abhängig von der Datei
→ Aufruf einer virtuellen Methode (objektorientiert)
- Verbindung zum Betriebssystemkern: System Call
Das Betriebssystem hat mehr Rechte als die Anwendung, daher darf die Anwendung Funktionen des Betriebssystems nur auf sehr kontrollierte Weise aufrufen.
→ Hardware-Unterstützung
Beispiel: x64-Befehlsarchitektur: `syscall`

3.5 Der Betriebssystemkern (Kernel)

- „Eingang“
Wo nimmt der Kernel den `syscall`-Befehl entgegen?
- „Ausgang“
Wo ruft der Kernel die Callback-Funktionen der Treibermodule auf?
`ksys_write()`
`vfs_write()`
- „Weg durch den Kernel“
Was passiert dazwischen?

4 Speicherverwaltung

4.1 Microcontroller

Speicherverwaltung „einfach so“

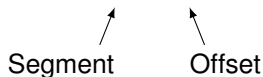
- PC erzeugt Speicher-Layout (z. B. Intel-[.hex](#)-Datei)
- Darin enthalten: Programm, Interrupt-Vektoren, ...
- Aufspielen in den Flash-Speicher

4 Speicherverwaltung

4.2 Speichersegmentierung

Intel 8086: Segment- und Offset-Adresse

- Beispiel: Speicherzelle **28F5:0100**: für das aktuelle Programm vorgesehen


Segment Offset

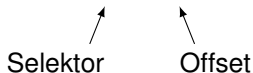
- Speicherzelle **B800:0000**: Bildschirmspeicher (Textmodus)
- physikalische Adresse = $0x10 \cdot \text{Segment} + \text{Offset}$, hier also: **B8000**
- Speicher insgesamt: 5 Hex-Ziffern = 20 Bit für Speicher-Adressierung
→ maximal 1 MiB adressierbar
- „Speicherschutz“: Das Betriebssystem weist jedem Programm ein Segment zu (hier z. B.: **28F5**). Außerhalb davon **sollte** man nicht schreiben.
(→ effiziente Bildschirmausgabe: direkt in Segment **B800** schreiben)

4 Speicherverwaltung

4.3 Speicherschutz (*Protected Mode*)

Intel 80286: Selektoren

- Beispiel: Speicherzelle **0007:0100**: für das aktuelle Programm vorgesehen


Selektor Offset

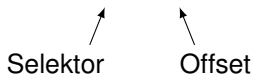
- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*
Segment-Deskriptor:
 - physikalische Adresse (z. B.: **28F5**) des Segments
 - Länge des Segments (maximal 64 kiB)
 - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- In der Praxis (MS-DOS): *kein* Speicherschutz, sondern:
Jedes Programm verwaltet die Deskriptorentabelle selbst.

4 Speicherverwaltung

4.3 Speicherschutz (*Protected Mode*)

Intel 80386: 32-Bit-Offsets

- Beispiel: Speicherzelle 0007:00000100



- Selektor = Index für ein Array innerhalb des Prozessors: *Deskriptorentabelle*
Segment-Deskriptor:
 - physikalische Adresse (z. B.: 28F5) des Segments
 - Länge des Segments (maximal 4 GiB)
 - Zugriffsrechte: Lesen, Schreiben, Ausführen→ Programme gegeneinander absichern (mit Hardware-Unterstützung)
- MS-DOS und MS-Windows bis Version ME: *kein* Speicherschutz, sondern: Jedes Programm verwaltet die Deskriptorentabelle selbst.
- MS-Windows ab Version NT, OS/2, MacOS, Linux und andere Unixe: Speicherschutz mit Hardware-Unterstützung

4 Speicherverwaltung

4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

- spezielle Variable („Register“) im Prozessor:
Berechtigungsstufe des aktuell laufenden Programms
- Benutzerprogramme dürfen Befehle zur Manipulation der Deskriptorentabelle nicht ausführen. Dies darf nur der Betriebssystemkern.
- Speicherzugriffe außerhalb des zugewiesenen Segments lösen eine *Exception* aus (ähnlich Interrupt). Das Betriebssystem kann daraufhin das Programm kontrolliert beenden („Speicherzugriffsfehler“).
- Das Betriebssystem kann Code als „nicht ausführbar“ markieren. Versuche, ihn auszuführen, lösen eine Exception aus.

4 Speicherverwaltung

4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren

4 Speicherverwaltung

4.3 Speicherschutz (*Protected Mode*)

Speicherschutz mit Hardware-Unterstützung

Beispiel: Puffer-Überlauf

- Die Rücksprung-Adresse (auf dem Stack) wird durch eine Variable (auf dem Stack) überschrieben
- Gezielter Sprung in die Variable hinein ist möglich.
offene Sicherheitslücke in alten Versionen von MS-Windows
- Lösung: Stack als „nicht ausführbar“ markieren
- Weiterhin möglich: *return-oriented programming*