Betriebssysteme
Übungen

Barry Linnert

Wintersemester 2017/18

Betriebssystementwicklung

- Betriebssystementwicklung ähnelt normaler Programmentwicklung
  - Algorithmen (z. B. Suchen, Sortieren)
  - Strukturen (z. B. Listen, Bäume, Arrays)
  - Pointerarithmetik, Bitmanipulation, . . .
  - Modularisierung, Design-Pattern, Objektorientierung, . . .
  - Bearbeitung, Versionsverwaltung, Tests, Debugging, . . .

- Aber es gibt auch Unterschiede
  - keine Hardwareabstraktion
  - keine Laufzeitumgebung
  - kein Netz, kein doppelter Boden
  - Tests und Debugging u. U. schwieriger/langwieriger

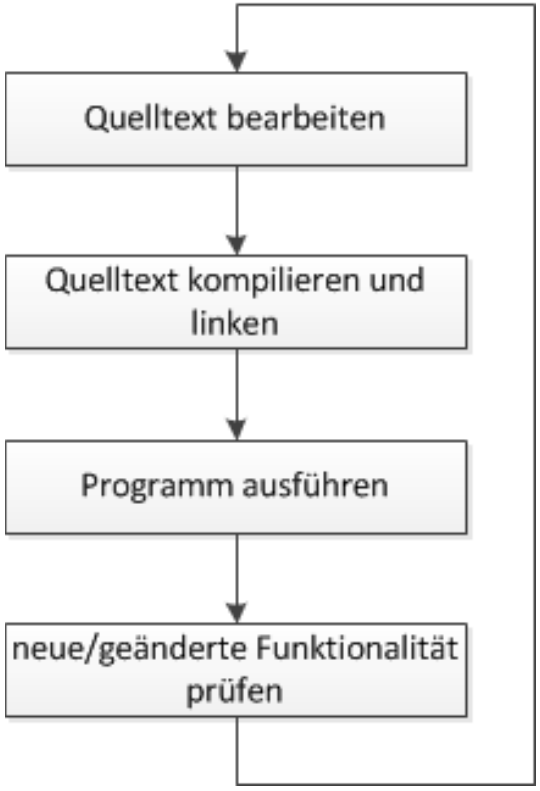
Prinzipielle Möglichkeiten der Entwicklung

Freie Universität Berlin

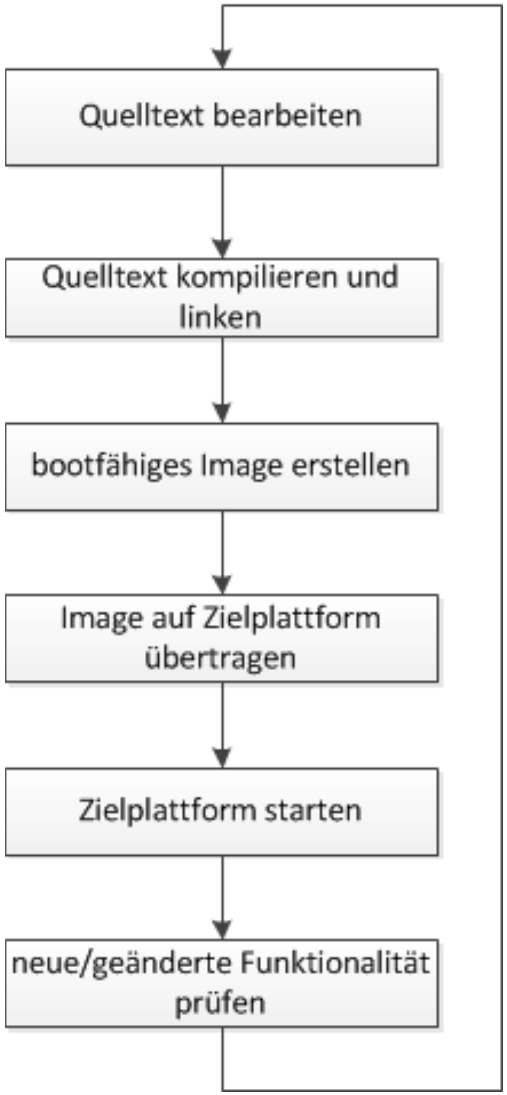
- Self-hosted
  - Entwicklungsumgebung läuft auf Zielsystem
  - Code wird auf gleichem System erzeugt, auf dem er ausgeführt wird oder ausgeführt werden kann
  - für Entwicklung normaler Anwendungen

- Cross-Compilation
  - Entwicklungsumgebung läuft auf anderem System
  - unterschiedliche Plattformen erfordern einen Cross-Compiler
  - erzeugter Code wird auf Zielplattform übertragen bzw. Zielplattform wird mit erzeugtem Code gestartet
  - für leistungsschwache oder anderweitig eingeschränkte Zielsysteme

Normaler Entwicklungszyklus



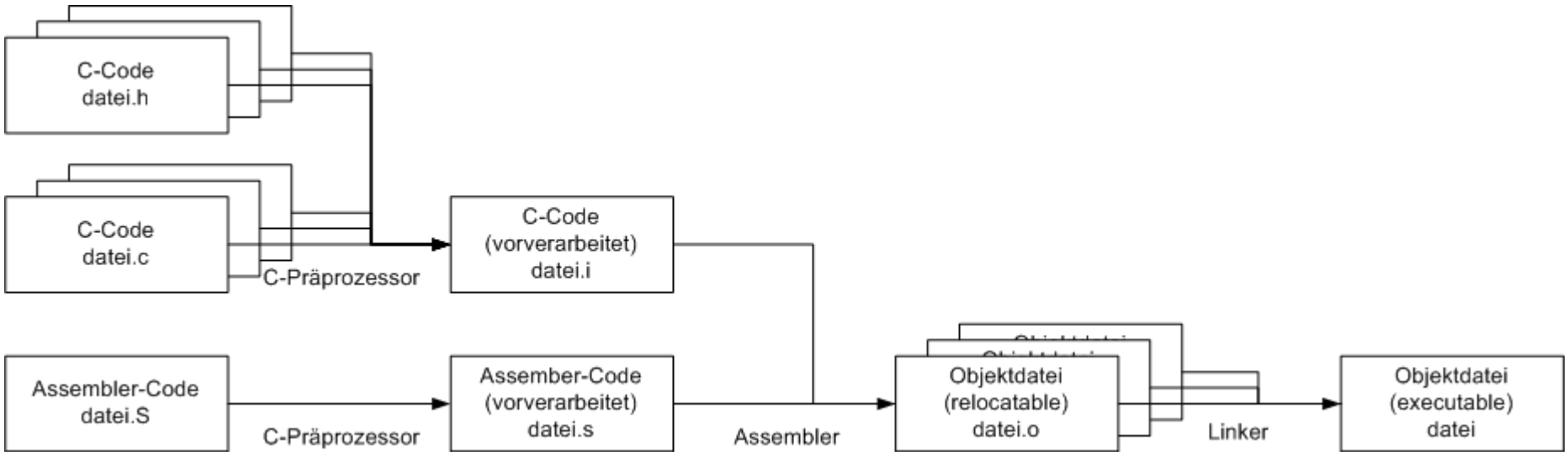
Betriebssystem-Entwicklungszyklus



Entwicklungsumgebung

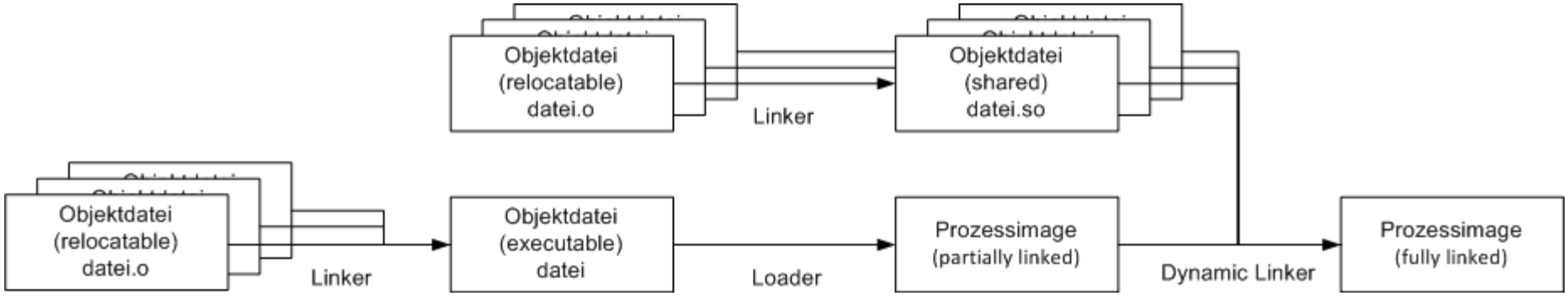
- GNU-Tools für die Entwicklung
  - `gcc` als Cross-Compiler
  - `binutils` zum Cross-Assemblieren und Linken
  - `make` zum Automatisieren von Entwicklungsvorgängen
- QEMU als Emulator für Zielplattform
  - Standard-QEMU mit eigens hinzugepatchter Unterstützung für Zielplattform
  - Abgaben müssen mittels toolchain im Linux-Pool laufen
- (Zielplattform mit U-Boot als Bootloader)

Vom Code zum Programm



- Objektdateien liegen im Executable and Linking Format (ELF) vor.

Vom Programm zum Prozess



- Anschließend kann der vom Programm vorgegebene Einsprungspunkt angesprungen werden.
- Bei statischem Linken entfällt der letzte Schritt.

Executable and Linking Format (ELF)

- ELF ist ein Format zur Beschreibung von Maschinencodeschnipseln
  - Code und Daten sind in Sektionen untergebracht
  - weitere (Meta-)Sektionen enthalten u. a. Symbol- und Relokationsinformationen
  - der ELF-Header enthält u. a. den Einsprungspunkt

- Jede Sektion verfügt u. a. über folgende Metadaten
  - Virtual Address (VMA): Code erwartet, dass Sektion sich zur Laufzeit an VMA befindet
  - Load Address (LMA, indirekt über Program Header): Loader soll Sektion an LMA laden
  - andere Metadaten, z. B. Schreibbar? Ausführbar? Null?

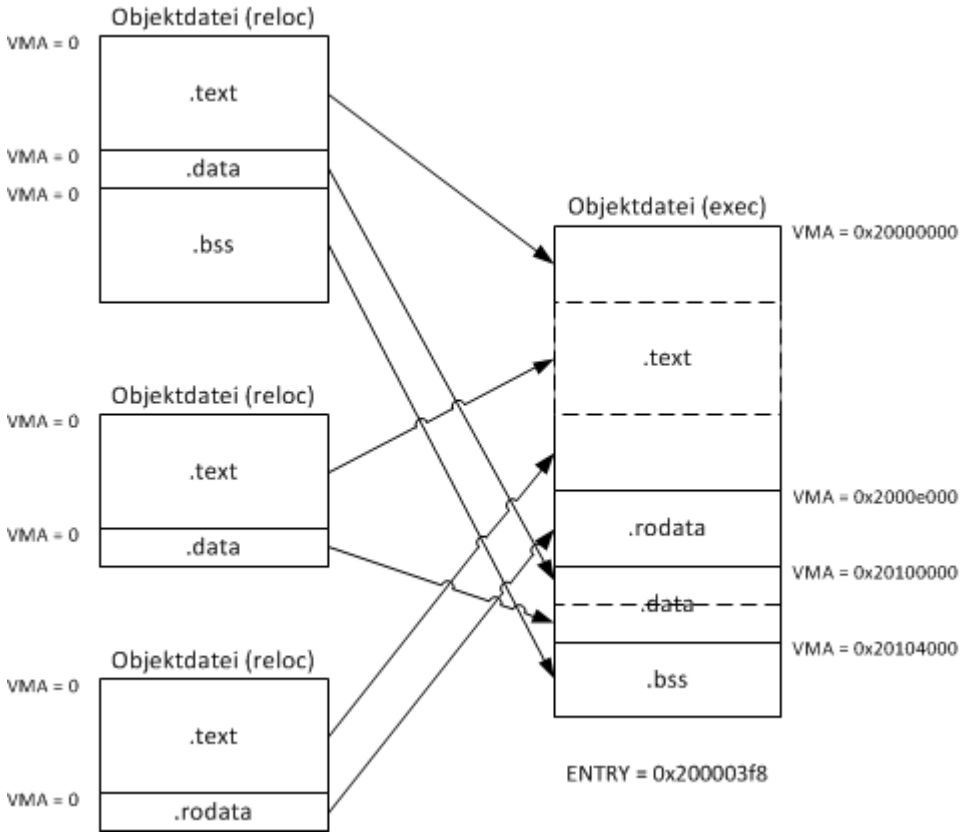
Executable and Linking Format (ELF)

- Symboltabelle enthält Namen und Adressen aller Funktionen, Variablen und vergleichbaren Dingen.

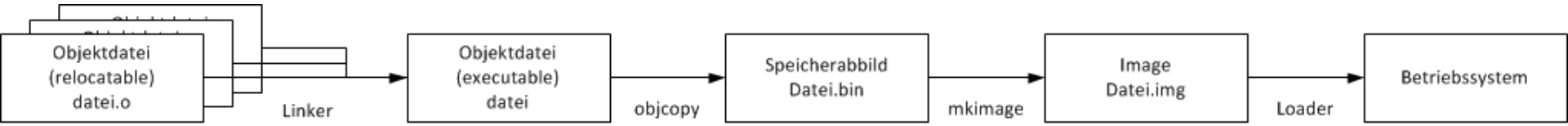
- Relokationsinformationen beschreiben sämtlichen Codestellen, die ein Symbol referenzieren
  - Welches Symbol wird referenziert?
  - Wo und wie wird es referenziert?

Executable and Linking Format (ELF)

- Typische Sektionen
  - `.text`: Code, nur lesbar
  - `.data`: initialisierte Daten
  - `.rodata`: Konstanten, nur lesbar
  - `.bss`: Nullen (nicht tatsächlich gespeichert)
- Beim Linken werden
  - Sektionen zusammengefasst
  - Adressen festgelegt/ver-schoben
  - Code und Daten anhand Symbol- und Relokationsinformationen angepasst (d. h. jetzt bekannte Adressen von Symbolen werden überall dort eingetragen, wo sie benötigt werden)



Vom Programm zum Betriebssystem



- Der Bootloader auf der Zielplattform, U-Boot, versteht nur relative einfache Images bestehend aus
  - Speicherabbild (ggf. komprimiert)
  - Ladeadresse
  - Einsprungspunkt
- Speicherabbild wird erstellt mit `objcopy` (gehört zu `binutils`).
- Ladeadresse und Einsprungspunkt werden mittels `mkimage` (gehört zu U-Boot) spezifiziert.

Entwicklungszyklus mit QEMU

- QEMU kann Dateien im ELF-Format laden; es muss kein Image erzeugt werden.