Hardwarenahe Programmierung I

U. Kaiser, R. Kaiser, M. Stöttinger, S. Reith

(HTTP: http://www.cs.hs-rm.de/~kaiser
EMail: robert.kaiser@hs-rm.de)

Wintersemester 2021/2022

1. Einführung

http://www.interaktiv-narrativ.org/media/vorspann.jpg

Was ist „hardwarenahes Programmieren“?

Hochschule RheinMain

- Z.B. Google-Suche liefert keine klare Definition des Begriffs
  (Stattdessen aber sehr viele interessante Stellenangebote...)
- Vorstellung / Kenntnis der Vorgänge innerhalb des Rechensystems beim Ausführen von (Hochsprache-)Programmen
  - Speicherorganisation (Code / Daten / Stack / Heap)
  - Aufrufschnittstelle (*Call by Value* / *Reference* und Konsequenzen)
  - Fallstricke (Pufferüberlauf, Stack-Überlauf, Nebenläufigkeit, Compileroptimierungen, Caches, virtuelle Adressierung...)
  - Tipps und Tricks: Effizienzoptimierung, Fehlersuche

→ „Blick über den Tellerand“ des Hochsprachen-Modells

- Jedes Rechensystem braucht zwingend hardwarenahe Programme
  (Betriebssysteme / Gerätetreiber / E/A-Bibliotheken)

→ Stark nachgefragte (aber selten anzutreffende) Fähigkeit

Maschinensprache

- Die Hardware führt Programme in *Maschinensprache* aus
- Vorteil: vollkommene Kontrolle
  - Maximale Effizienz (?)
  - Größte Freiheiten in der Wahl des Programmiermodells
  - Zugriff auf Besonderheiten der Architektur (Register, Port-Mapped I/O, etc.)
- Nachteile:
  - Höchster Anspruch an Fähigkeiten und Kenntnisse der ProgrammiererInnen
  - Programme (und Kenntnisse) sind nicht „portabel“
  - Komplexe Algorithmen kaum in Maschinensprache beherrschbar
  - Fehlerträchtig: z.B. kein Typkonzept
- Heute wird nur noch in seltenen Ausnahmefällen in Maschinensprache programmiert (i.d.R. um Dinge zu tun, die in der Hochsprache „nicht gehen“, z.B. Interrupts maskieren, auf bestimmte Register zugreifen, etc.)

Warum C?

Hochschule RheinMain

- C ist eine portable Programmiersprache, d.h. C-Programme können Im Prinzip$^{\mathrm{TM}}$auf jeder Maschine laufen, für die es einen C-Compiler gibt
- Dennoch sind typische maschinenspezifische Konzepte wie Stack, Speicheradressen, etc. in C noch zugänglich
- Sprache ist unabhängig von Laufzeitbibliothek
- Es gibt eine exakt definierte Schnittstelle (*Application Binary Interface*, ABI) zum Maschinencode
- Dadurch können aus C heraus problemlos Maschinencode-Prozeduren gerufen werden (und umgekehrt)
- Zudem können bei den meisten C-Compilern Assemblerbefehle direkt in C-Code eingebettet werden
  (*inline assembler)*

Themen der Vorlesung

- Technische Grundlagen:
  Mini-Einblick Rechnerarchitektur, CPU, Assembler
- Abbildung Assembler $\leftrightarrow$ C:
  Statements, Variablen, Prozeduren
- C als Sprache:
  Keywords, Syntax, Module, Kontrollstrukturen, Statements,
  Variablen: Geltungsbereich, Typen
- Hardwaremäßige Repräsentierung von Variablen:
  `volatile`, `struct`, `union`, Portabilität
- Compiler, Build-Umgebungen (`make`, IDEs)
- Debugging
- Standardbibliotheken: Übersicht

Konkret: Kapitel der Vorlesung

Hochschule RheinMain

1. Einführung
2. Programmierung allgemein
3. Programmieren auf Maschinenebene
4. Hochsprachen
5. C-Sprachelemente
6. Arithmetik
7. Modularisierung
8. Zeiger und Adressen
9. Programmstruktur
10. C-Bibliothek
11. Datenstrukturen