Systemnahe Programmierung in C (SPiC)

36 Speicherorganisation – Zusammenfassung

Jürgen Kleinöder, Daniel Lohmann, Volkmar Sieh

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2020

http://www4.cs.fau.de/Lehre/SS20/V_SPIC

Statische versus dynamische Allokation

- Bei der **μC-Entwicklung** wird statische Allokation bevorzugt
  - **Vorteil:** Speicherplatzbedarf ist bereits nach dem Übersetzen / Linken exakt bekannt (kann z. B. mit `size` ausgegeben werden)
  - Speicherprobleme frühzeitig erkennbar (Speicher ist knapp! ↪ 1–4)

```
~> size sections.avr
text    data    bss    dec    hex filename
682     10      6      698    2ba sections.avr
```

Sektionsgrößen des Programms von ↪ 34–1

⇝ Speicher möglichst durch `static`-Variablen anfordern

- Regel der geringstmöglichen Sichtbarkeit beachten ↪ 12–6
- Regel der geringstmöglichen Lebensdauer „sinnvoll“ anwenden

- Ein Heap ist verhältnismäßig teuer ⇝ wird möglichst vermieden
  - Zusätzliche Speicherkosten durch Verwaltungsstrukturen und Code
  - Speicherbedarf zur Laufzeit schlecht abschätzbar
  - Risiko von Programmierfehlern und Speicherlecks

Statische versus dynamische Allokation (Forts.)

- Bei der Entwicklung für eine **Betriebssystemplattform** ist dynamische Allokation hingegen sinnvoll
  - **Vorteil:** Dynamische Anpassung an die Größe der Eingabedaten (z. B. bei Strings)
  - Reduktion der Gefahr von *Buffer-Overflow*-Angriffen

$\rightsquigarrow$ Speicher für Eingabedaten möglichst auf dem Heap anfordern

- Das Risiko von Programmierfehlern und Speicherlecks bleibt!