

# Verlässliche Echtzeitsysteme

## Übungen zur Vorlesung

Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)  
<https://www4.cs.fau.de>

Sommersemester 2019



- 1 C-Quiz Teil VI
- 2 Stack- & Laufzeitanalyse
- 3 Aufgabenstellung



**1** C-Quiz Teil VI

2 Stack- & Laufzeitanalyse

3 Aufgabenstellung



- C99
- x86 bzw. x86-64, d. h.
  - vorzeichenbehaftete Integer als Zweierkomplement implementiert
  - char hat 8 Bit
  - short hat 16 Bit
  - int hat 32 Bit
  - long hat 32 Bit auf x86 und 64 Bit auf x86-64



## Frage 17

Angenommen  $x$  hat Typ `int`. Ist  $x - 1 + 1 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von  $x$ ?

### Erklärung

- additive Operatoren sind linksassoziativ
- ⇒ nicht definiert für `INT_MIN`



## Frage 18

Angenommen  $x$  hat Typ `int`. Ist `(short)x + 1 ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von  $x$ ?

### Erklärung

- wenn  $x$  nicht in `short` passt

↪ Verhalten ist implementierungsabhängig



## Frage 19

Angenommen  $x$  hat Typ `int`. Ist `(short) (x + 1) ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von  $x$ ?

### Erklärung

- wenn  $x + 1$  nicht in `short` passt
  - ↳ implementierungsabhängig
- die meisten Compiler schneiden beim Cast ab



## Frage 20

Hat die Auswertung von `INT_MIN % -1` definiertes Verhalten?

1. ja
2. nein
3. das weiß niemand ...

### Erklärung

- C99 macht dazu keine Aussage
- in C11 gilt folgendes:
  - wenn  $(a/b)*b + a\%b$  darstellbar ist, haben  $a/b$  und  $a\%b$  definiertes Verhalten
  - sonst nicht
  - $\text{INT\_MIN} / -1$  entspricht  $\text{INT\_MAX} + 1$
  - was auf x86/x86-64 nicht darstellbar ist





1 C-Quiz Teil VI

**2** Stack- & Laufzeitanalyse

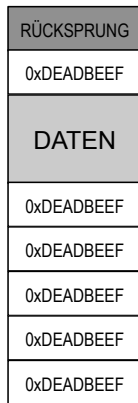
3 Aufgabenstellung





- Harte, verlässliche Echtzeitsysteme
  - Garantien über Ressourcenbedarf notwendig
    - ☞ statische Analyse unabdingbar
- Mögliche Ressourcen: Speicherbedarf, Laufzeit, etc.
- Übung: Analyse des Stackverbrauchs einer Feldbibliothek
- Stack-Analyse
  1. Dynamisch: Wasserstandstechnik
  2. Statisch: „Eigenbau“ und aiT (Stack-Analyzer der a<sup>3</sup> Suite)
- WCET-Analyse mittels aiT (bereits in EZS behandelt)

- **Messung zur Laufzeit:** Water-Marking (siehe Vorlesung)
- Grundidee: Einfügen von **Stack Canaries**
- Explizite Verwaltung des Stapelspeichers notwendig
- pthread-Bibliothek ermöglicht Verwaltung
- Mögliche Canaries
  - Lesbare Bitmuster: 0xDEADBEEF
  - Unwahrscheinliche Bitmuster: 0b101010101010...





## 1. (Globalen) Stack anlegen:

```
1 static unsigned int g_data[DATA_SIZE];
```

## 2. Thread anlegen & starten:

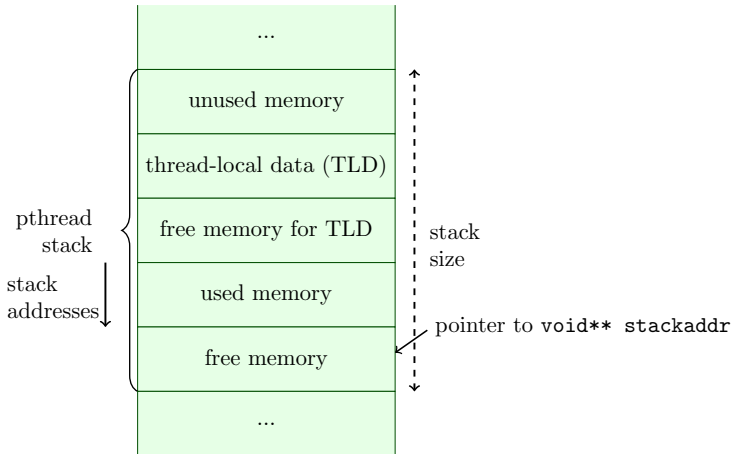
```
1 pthread_t thread;  
2 pthread_attr_t attr;  
3 pthread_attr_init(&attr);  
4 pthread_attr_setstack(&attr, &g_stack, STACK_SIZE);  
5 // worker function: void *run(void *param)  
6 int status = pthread_create(&thread, &attr, run, NULL);  
7 if (status != 0) { ... // handle error
```

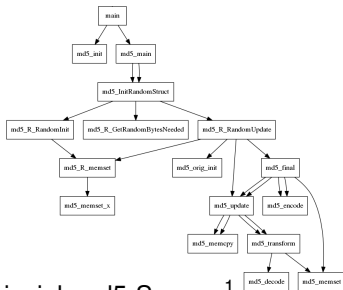
## 3. Auf Thread warten:

```
1 pthread_join(thread, &ret);
```



# pthread Stack





```

1  /* Objective function */
2  max: +16 md5_orig_init +64 md5_updat
3      +16 md5_memset +208 md5_transform
4
5
6  /* Constraints */
7  +main = 1;
8  +md5_init +md5_main <= +main;
9  ...
  
```

■ Beispiel: md5-Summe<sup>1</sup>

■ Vorgehen

1. Callgraph bestimmen
2. Stackverbrauch einzelner Funktionen (gcc -fstack-usage)
3. ILP<sup>2</sup> aufstellen (Nebenbedingungen aus 1., Kosten aus 2. verwenden)
4. ILP z.B. mittels lp\_solve  $\rightsquigarrow$  **worst-case Stackverbrauch**

<sup>1</sup><https://github.com/tacle/tacle-bench/>

<sup>2</sup>Integer Linear Program (dt. ganzzahliges lineares Programm)

## Optimierungsziel

- Jeder Stapelrahmen einer Funktion  $f$  hat eine Größe  $size$
- Jede Funktion kann auf einem Pfad ein- oder mehrfach (Rekursion), insgesamt  $n$ -fach auf dem Stapel vorkommen
- Gesucht ist Pfad durch den Aufrufgraphen, welcher Stapelbedarf maximiert
- Dabei müssen **Flussbedingungen** eingehalten werden
  - Aufruferbeziehung
  - Alternativen
  - ...

## Optimierungsziel

$$\max \sum_{\text{Funktion } f} size_f \cdot n_f$$

In lp\_solve -Syntax: `max : +64 n_f1 +48 n_f2 +42 n_f3 ;`



# Flussbedingung: Initialer Aufruf

## Semantik

Der initiale Aufruf erfolgt maximal (wahlweise auch genau) ein mal

## Formalisierung

$$n_{\text{main}} \leq 1$$



## lp\_solve -Syntax

```
n_main <= 1;
```





# Flussbedingung: Mehrere Vorgänger

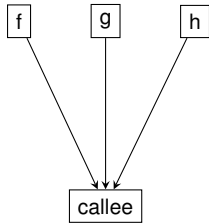
## Semantik

Jede Funktion kann nur so oft ausgeführt werden, wie sie von den Vorgängern aus aufgerufen wird

## Formalisierung

Sei  $f_{a \rightarrow b}$  die Anzahl der Aufrufe von  $b$  durch  $a$ :

$$n_{callee} \leq \sum_{p \in \text{Aufrufer}(callee)} f_{p \rightarrow callee}$$



## lp\_solve -Syntax

```
n_caller <= + f_f_callee + f_g_callee + f_h_callee ;
```



# Flussbedingung: Immer nur ein Nachfolger pro Funktion

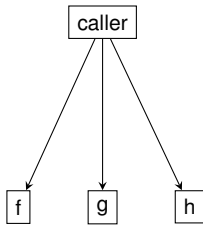
## Semantik

Jede Funktionsinkarnation ruft gleichzeitig jeweils maximal eine weitere Funktion auf

## Formalisierung

Sei  $f_{a \rightarrow b}$  die Anzahl der Aufrufe von  $b$  durch  $a$ :

$$\sum_{c \in \text{Aufgerufene}(\text{caller})} f_{\text{caller} \rightarrow c} \leq n_{\text{caller}}$$



## lp\_solve -Syntax

```
+ f_caller_f + f_caller_g + f_caller_h <= n_caller ;
```



# Flussbedingung: Rekursion

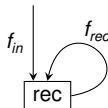
## Semantik

Rekursive Funktionen können pro Aufruf von außen bis zu ihrer maximalen Rekursionstiefe ( $d$ ) oft ausgeführt werden.

## Formalisierung

$$f_{rec} \leq d_{rec} \cdot f_{in}$$

$$n_{rec} \leq f_{in} + f_{rec}$$



## lp\_solve -Syntax

```
f_rec <= +42 f_in ;  
n_rec <= f_in + f_rec ;
```



# Beispiel

- Problemformulierung in lpsolve:

```
max: +40 n_main +20 n_f +60 n_g;
```

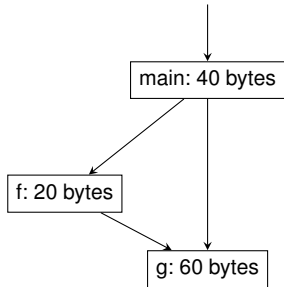
```
n_main <= 1;  
+f_main_f +f_main_g <= n_main;  
n_f <= +f_main_f;  
+f_f_g <= n_f;  
n_g <= +f_f_g +f_main_g;
```

- Ausgabe von lp\_solve :

```
Value of objective function: 120.00000000
```

```
Actual values of the variables:
```

n_main	1
n_f	1
n_g	1
f_main_f	1
f_main_g	0
f_f_g	1



```
$ lp_solve infeasible.lp  
This problem is infeasible
```

### Infeasible Models

Eine oder mehrere der Nebenbedingungen führen zu einem logischen Widerspruch

Leider bietet `lp_solve` selbst direkt keine Hilfestellung zur Lokalisation. Die Entwickler empfehlen das Einführen von “slack”-Variablen:<sup>3</sup>

<code>max: x + y;</code>	<code>max: x + y;</code>	<code>x: 20</code>
<code>x + 1 &lt;= x;</code>	<code>x + 1 - e_1 &lt;= x;</code>	<code>y: 20</code>
<code>y &gt; y + 1;</code>	<code>y + e_2 &gt; y + 1;</code>	<code>e_1: 1</code>
<code>x &lt;= 20;</code>	<code>x &lt;= 20;</code>	<code>e_2: 1</code>
<code>y &lt;= 20;</code>	<code>y &lt;= 20;</code>	

<sup>3</sup><http://lpsolve.sourceforge.net/5.5/Infeasible.htm>

```
$ lp_solve unbounded.lp  
This problem is unbounded
```

## Unbounded Models

Eine oder mehrere der Variablen sind nach oben unbeschränkt

Durch künstliche Beschränkung aller Variablen im System (auf einen sehr großen Wert) lassen sich unbeschränkte Variablen detektieren:

max: $x + y + z$ ;	max: $x + y + z$ ;	x: 5000
$z \leq y + 1$ ;	$z \leq y + 1$ ;	y: 20
$y \leq 20$ ;	$y \leq 20$ ;	z: 21
	$x \leq 5000$ ;	
	$y \leq 5000$ ;	
	$z \leq 5000$ ;	



- `lp_solve` ist auf die Lösung linearer Gleichungssysteme ausgelegt
- Es ist dementsprechend nicht möglich, zwei Variablen zu multiplizieren
  - `a * b`  $\Rightarrow$  Syntaxfehler
  - `max : a b`  $\Rightarrow$  optimiert  $a + b$
- Lösung in VEZS für Konstanten (Stapelrahmengrößen): C-Präprozessor:

```
#define s_main 40  
#define s_f    20  
#define s_g    60
```

```
max: +s_main n_main +s_f n_f +s_g n_g;
```

~> `stackusage / lp_solvepp`



- Statische Code-Analyse mit a<sup>3</sup> Tool-Suite
  1. aiT: WCET-Analyse
  2. Stack-Analyzer: Stackverbrauch
  3. ...
- Installiert im CIP-Pool
- Verfügbare Rechner:
  - CIP2: faui00a-y
  - CIP2: faui02a-y
  - CIP2: faui0fa-u





# a<sup>3</sup> Analyzer – Neue Projekt Anlegen

The screenshot shows the 'New' menu of the a3 Analyzer. The menu items are:

- Open... (Ctrl+O)
- Recently opened
- Save project
- Save as project...
- Save as workspace...
- Import project
- Compare projects...
- Start all enabled analyses (F6)
- Start analysis (F7)
- Start analysis interactive (F8)
- Stop analysis (F4)
- Preferences... (Ctrl+,)
- Quit (Ctrl+Q)

The main window displays a list of analysis tools:

- ValueAnalyzer**: Program value analysis
- ResultCombinator**: Combination of results according to formula
- Control-Flow Visualizer**: Visualization of control-flow graph

At the bottom of the window, there are buttons for 'Search' and 'Memory usage'.



# a<sup>3</sup> Analyzer – Executable Angeben

The screenshot displays the a<sup>3</sup> Analyzer interface. On the left, a sidebar contains navigation options: Home, Analyses (with sub-items: Hardware, Reporting, Decoding, Stack & value analysis, Timing analysis, Source files, Configurations, Support), Setup, and Information. The main window is titled 'Files' and shows a large red rectangular area labeled 'Executables:'. A tooltip 'Browse for item+' is visible over this area. Below the red area are three input fields: 'AIS file:', 'Report file:', and 'XML results file:'. At the bottom, the 'Project:' field shows 'Temporary directory: /tmp/a3arm-psjBMK'. A search bar at the bottom left contains the text 'Filter...'. At the bottom right, there are 'Search' and 'Memory usage' buttons.



# a<sup>3</sup> Analyzer – Annotations-Datei Anlegen

The screenshot shows the a<sup>3</sup> Analyzer interface. On the left is a sidebar with a tree view containing the following items:

- Home
- Files
- stackanalyzer\_arm.ais
- Hardware
- Reporting
- Decoding
- Analyses
  - Stack & value analysis
  - Timing analysis
- f(x)
  - Source files
  - Configurations
  - Support
- Setup
- Information

At the bottom of the sidebar is a 'Filter...' dropdown menu.

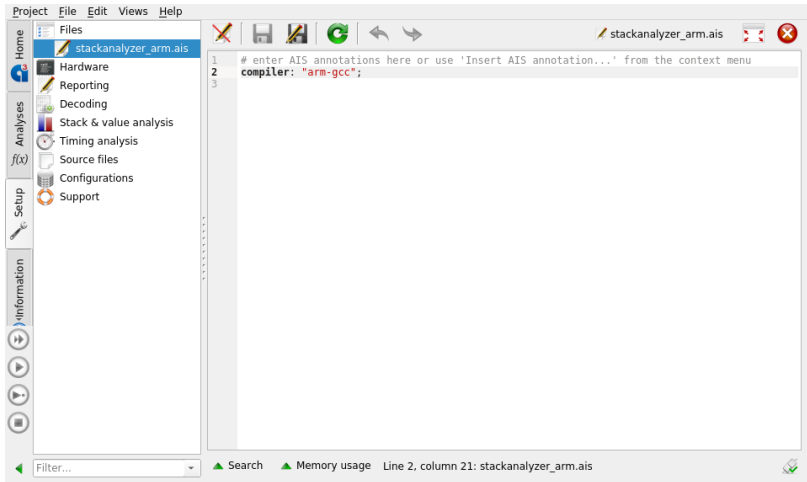
The main window has a title bar 'Files' and a close button. The 'Executables:' list contains one entry: 'lehre/vezs-aufgaben/06\_Analyse/build-arm/stackanalyzer\_arm'. Below this list are three input fields:

- AIS file: stackanalyzer\_arm.ais
- Report file: (empty)
- XML results file: (empty)

Each input field has a file selection icon on the right. There is an 'Edit AIS file' button next to the Report file field. At the bottom of the main window, the 'Project:' section shows 'Temporary directory: /tmp/a3arm-psjBMK'. There are also 'Search' and 'Memory usage' buttons at the bottom left, and a green checkmark icon at the bottom right.



# a<sup>3</sup> Analyzer – Compiler-Annotation Anlegen



The screenshot displays the a3 Analyzer IDE interface. The top menu bar includes 'Project', 'File', 'Edit', 'Views', and 'Help'. The toolbar contains icons for file operations (delete, save, copy, paste, undo, redo) and window management (maximize, close). The left sidebar shows a project tree with categories: Home (Files, stackanalyzer\_arm.ais), Hardware, Reporting, Decoding, Analyses (Stack & value analysis, Timing analysis), f(x) (Source files, Configurations), Setup (Support), and Information. The main editor window shows the file 'stackanalyzer\_arm.ais' with the following content:

```
1 # enter AIS annotations here or use 'Insert AIS annotation...' from the context menu
2 compiler: "arm-gcc";
3
```

The status bar at the bottom indicates 'Line 2, column 21: stackanalyzer\_arm.ais' and includes search and memory usage icons.



# a<sup>3</sup> Analyzer – Stack-Analyse Selektieren

The screenshot shows the a3 Analyzer application window. The menu bar includes 'Project', 'Views', and 'Help'. The 'Project' menu is open, showing 'f(x) Create'. The main window title is 'f(x)'. Below the menu bar, there is a vertical toolbar with icons for 'Home', 'Analyses', 'Setup', and 'Information'. The 'Analyses' section is active, displaying a list of analysis tools:

- aiT**: Safe WCET analysis (represented by a clock icon)
- StackAnalyzer**: Stack usage analysis (represented by a bar chart icon)
- TimeWeaver**: Trace-based WCET estimation (represented by a clock and trace icon)
- ValueAnalyzer**: Program value analysis (represented by a binary code icon)
- ResultCombinator**: Combination of results according to formula (represented by a chalkboard icon)
- Control-Flow Visualizer**: Visualization of control-flow graph (represented by a flowchart icon)

At the bottom of the window, there is a 'Filter...' dropdown menu and two buttons: 'Search' and 'Memory usage'.



# a<sup>3</sup> Analyzer – Stack-Analyse Starten

The screenshot displays the a3 Analyzer interface. At the top, there is a menu bar with 'Project', 'Analysis', 'Views', and 'Help'. Below the menu bar is a toolbar with icons for file operations and analysis. The main window is titled 'StackAnalyzer' and contains the following fields:

- ID: StackAnalyzer
- Comment: (empty)
- Result: n/a

Below these fields are two tabs: 'Settings' and 'Output'. The 'Settings' tab is active and contains the following configuration options:

- Configuration: Default Configuration
- Dependencies: (empty)
- Analysis start: main
- AIS file: (empty)
- Expected result: (empty)

At the bottom of the window, there is a status bar with a 'Filter...' dropdown, and three status indicators: 'Messages', 'Search', and 'Memory usage'. A small green checkmark icon is visible in the bottom right corner of the status bar.



# a<sup>3</sup> Analyzer – Stack-Analyse Starten

The screenshot shows the StackAnalyzer application window. The title bar includes the menu items "Project", "Analysis", "Views", and "Help". The main window is titled "StackAnalyzer" and contains the following elements:

- Left sidebar:** A vertical navigation pane with buttons for "Home", "Analyses", "Setup", and "Information". The "Analyses" section is active, showing a list of analyses with a "Start all enabled analyses" button highlighted in yellow.
- Top toolbar:** Contains icons for file operations and a ".ais" file icon.
- Main content area:** A form for configuring an analysis.
  - ID:** StackAnalyzer
  - Comment:** (empty text field)
  - Result:** n/a
  - Settings / Output tabs:** The "Settings" tab is selected.
    - Configuration:** Default Configuration (dropdown menu)
    - Dependencies:** (dropdown menu)
    - Analysis start:** main (text field with a green checkmark icon)
    - AIS file:** (text field with a folder icon and a yellow pencil icon)
    - Expected result:** (text field)
- Bottom status bar:** Includes a "Filter..." dropdown, and icons for "Messages", "Search", and "Memory usage".

1 C-Quiz Teil VI

2 Stack- & Laufzeitanalyse

**3 Aufgabenstellung**





- Existierende Implementierung: Array-Datenstruktur
- Vorgegebene Funktionen: Sortieren, Maximumssuche, ...
- Aufgaben
  1. Dynamische Analyse
    - 1.1 Thread erstellen
    - 1.2 Stack initialisieren
    - 1.3 Programm (mit Eingabedaten) ausführen
    - 1.4 Stackverbrauch messen
  2. Statische Analyse
    - 2.1 ILP aus Aufrufgraph aufstellen
    - 2.2 Mittels `lp_solve` lösen
    - 2.3 Optional: Verwendung `a3` Stack-Analyzer



Fragen?

