Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Analyse

Tobias Klaus, Florian Schmaus, Peter Wägemann

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

20. Juni 2016

Überblick

Annahmen

- 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` ...

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
  - ⇝ 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
  - `INT_MIN / -1` entspricht `INT_MAX + 1`
  - was auf x86/x86-64 nicht darstellbar ist

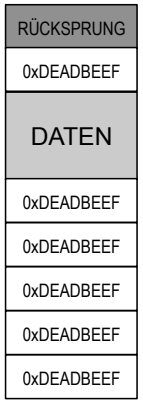
Überblick

Analyse Harter Echtzeitsysteme

- Harte, verlässliche Echtzeitsysteme
  - Garantien über Ressourcenbedarf notwendig
    ☞ statische Analyse unabdingbar
- Mögliche Ressourcen: Speicherbedarf, Laufzeit, etc.
- Übung: statische Analyse des Programms
- Stack-Analyse (StackAnalyzer der a3 Suite):
  1. Wasserstandstechnik
  2. aiT
- WCET-Analyse mittels aiT (bereits in EZS behandelt)

Dynamische Analyse des Stapelspeicherverbrauchs

- Messung zur Laufzeit: Water-Marking (siehe Vorlesung)
- Grundidee: Einfügen von Stack Canaries
- Explizite Verwaltung des Stapelspeichers notwendig
- pthread-Bibliothek ermöglicht Verwaltung (siehe Abschnitt 1)
- Mögliche Canaries
  - „Lesbare" Bitmuster: `0xDEADBEEF`
  - Unwahrscheinliche Bitmuster: `0b101010101010...`



pthread-Bibliothek

1. (Globalen) Stack anlegen:

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

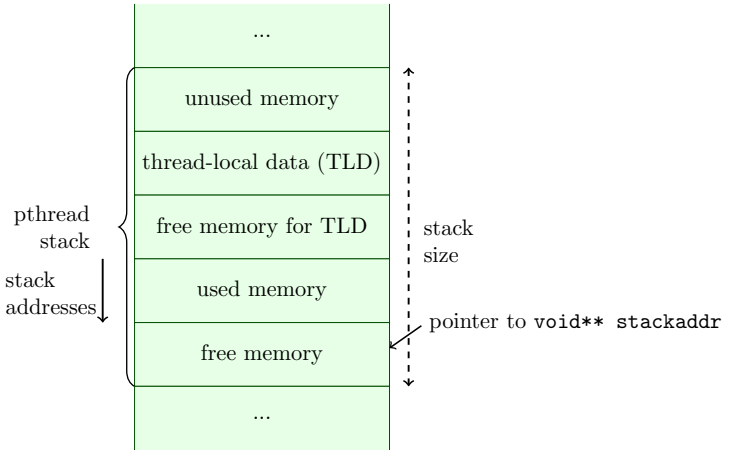
2. Thread anlegen & starten:

```
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);
```

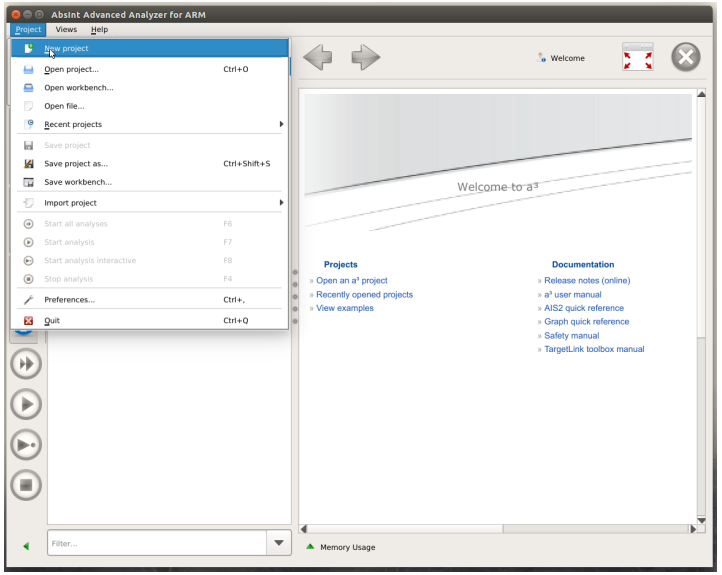
pthread Stack



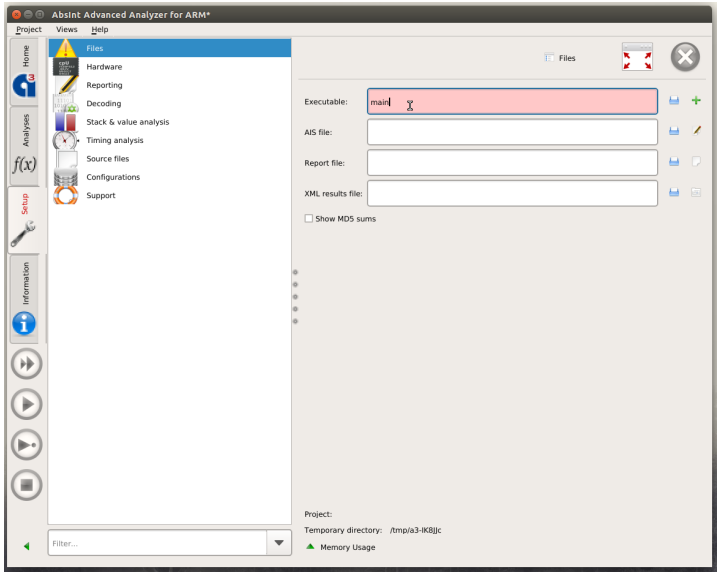
Statische Analyse des Stackverbrauchs

- Statische Code-Analyse mit a^3 Tool-Suite
  1. aiT: WCET-Analyse
  2. StackAnalyzer: Stackverbrauch
  3. ...
- Installiert im CIP-Pool
- Verfügbare Rechner:
  - 00.153-113 CIP3: faui0da-u
  - 00.156-113 CIP4: faui0ca-q

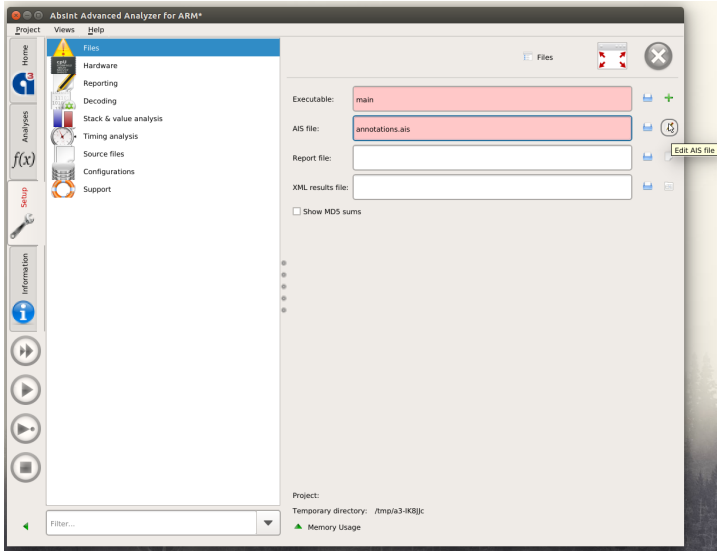
a^3 Analyzer – Neue Projekt Anlegen



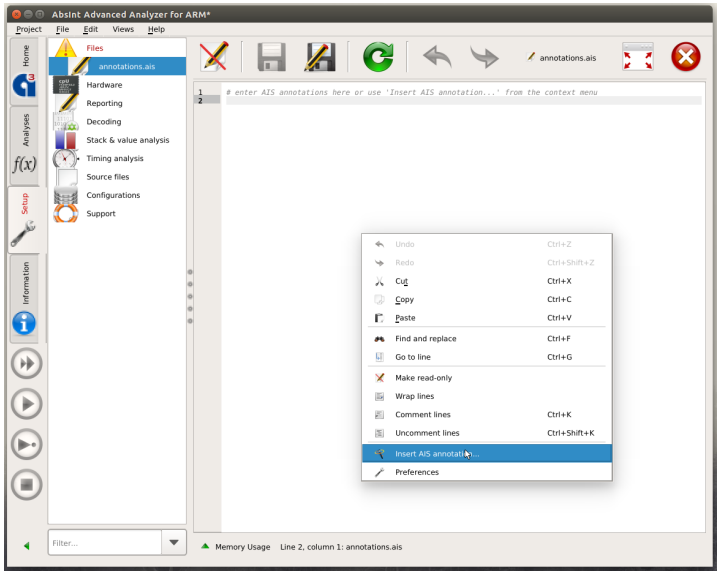
a^3 Analyzer – Executable Angeben



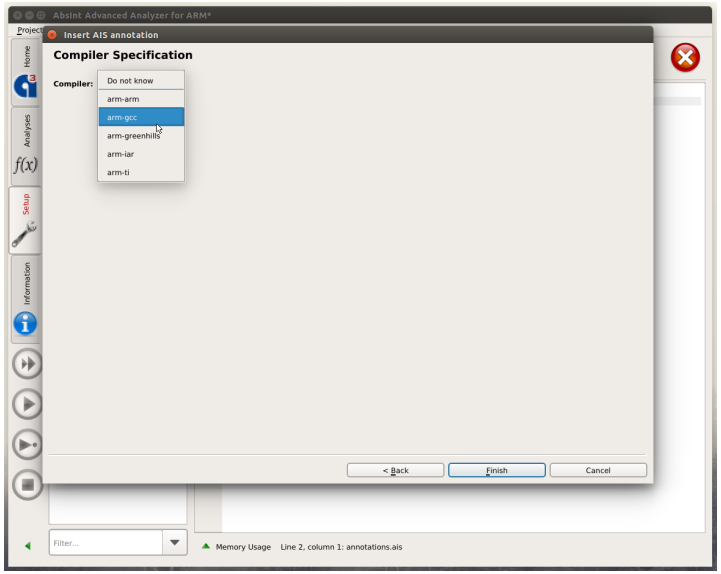
a^3 Analyzer – Annotations-Datei Anlegen



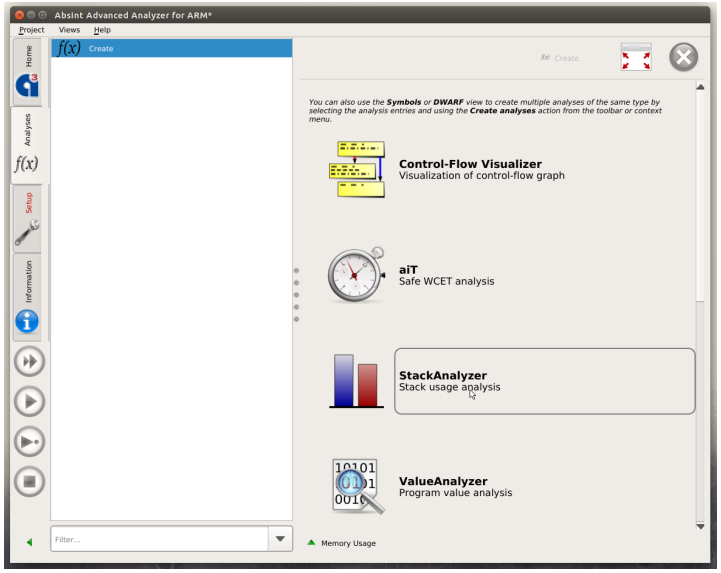
a^3 Analyzer – Compiler-Annotation Anlegen



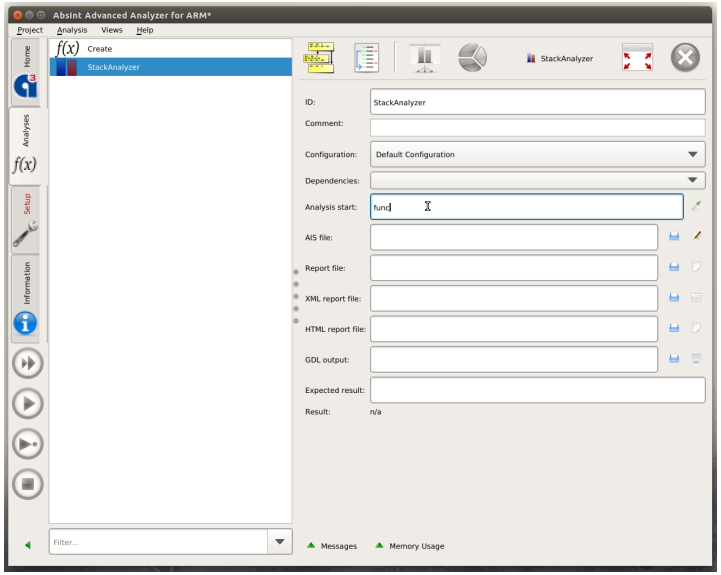
a^3 Analyzer – arm-gcc Angeben



a^3 Analyzer – Stack-Analyse Selektieren



a^3 Analyzer – Stack-Analyse Starten



Überblick

Aufgabenstellung

- Existierende Implementierung: Baumstruktur
- Vorgegebene Funktionen: Einfügen, Ausgeben, Suchen, ...
- Aufgaben

1. Dynamische Analyse
   - 1.1 Thread erstellen
   - 1.2 Stack initialisieren
   - 1.3 Programm (mit Eingabedaten) ausführen
   - 1.4 Stack verbrauch messen
2. Statische Analyse
   - 2.1 Verwendung a3 StackAnalyzer
   - 2.2 Einfügen von Annotationen wo notwendig

Fragen?