AUFGABE 6: LAUFZEIT UND STACKVERBRAUCHSANALSYSE

In dieser Aufgabe werden Sie lernen, wie Sie Eigenschaften wie Stackverbrauch und Laufzeit sowohl dynamisch als auch statisch analysieren können. Zunächst werden Sie mit Hilfe der *Wasserstands*-Technik den Stapelspeicherverbrauch einer von uns gegebenen Bibliothek messen. Anschließend werden Sie das statische Analysewerkzeug `StackAnalyzer` der Firma AbsInt nutzen um den maximalen Stackverbrauch zu bestimmen. Auf freiwilliger Basis können sie sich zusätzlich mit dem Werkzeug `aiT` für die Wort-Case-Execution-Times(WCET)-Analyse vertraut machen. **Hinweis:** Es bietet sich an, Arbeitspakete für die einzelnen Gruppenteilnehmer zu vereinbaren.

Grundlegende Übung

Wasserstand

1. Vorgabe: Machen Sie sich mit unseren Vorgabe in `src/tree.c`, `include/tree.c` und `src/stackanalyzer.c` vertraut. *Was wurde von uns implementiert? Was leisten die Funktionen* `insert()`*,* `find()`*,* `find_maximum()` *und* `walk_tree()`*? Wo liegt der Stapelspeicher für den Faden der mit der Funktion* `run()` *gestartet wird?*

.....

Wir haben versucht Ihnen so viel Ärger wie möglich mit der `pthreads`-Bibliothek zu ersparen. Dennoch ist es eine gute Idee sich mit den Handbuchseiten von `pthread_create` und `pthread_attr_setstack` auseinanderzusetzen.

☞ `man 3p pthread_create`
☞ `man 3p pthread_attr_setstack`

2. Messaufbau: Implementieren Sie die noch fehlenden Komponenten für die Messung des Speicherverbrauchs gemäß der in der Vorlesung und Übung vorgestellten Methode. Hierzu gehören hauptsächlich Funktionen zum Initialisieren des Stapelspeichers und zur Auswertung des bisher maximal genutzten Stapelspeichers.

3. Messung: Messen Sie den Stack-Verbrauch der einzelnen von uns vorgegebenen Funktionen mit den von uns vorgegebenen Daten durch eine konkrete Ausführung des Programms. *Für welche Eingabewerte der Funktion* `find` *ergibt sich ihr höchste Stapelspeicherverbrauch?*

☞ `./stackanalyzer_x86`

.....
.....

Überlegen Sie sich eigene Eingabewerte für die Datenstruktur, die den Speicherver-

brauch maximieren. *Für welche Eingabesequenzen ergeben sich die höchste Speicherverbräuche für die einzelnen Funktionen?*

4. Optimierung: Unsere Implementierung verschwendet absichtlich Stapelspeicher. *An welchen Codestellen liegt das?* Reimplementieren Sie unsere Vorgabe an den entscheidenden Stellen und messen Sie nun den Stapelspeicherverbrauch ihrer verbesserten Version. *Wie viel weniger Stapelspeicher verbraucht Ihre Verbesserung in den in der vorherigen Teilaufgabe identifizierten schlimmsten Fällen?*

.....

Statische Stackanalyse

5. Compiler: Das Stackverbrauchsanalsysewerkzeug `StackAnalyzer` operiert direkt auf ausführbaren Dateien. Leider wird dieses Werkzeug nicht für die Intel-Architektur sondern nur eingebettete Architekturen entwickelt. Deshalb müssen Sie für diese Aufgabe einen Cross-Compiler für unterstütze ARM-Cortex-M4-Architektur verwenden. Dies erreichen Sie, indem Sie `cmake` wie folgt aufrufen:
`cmake -DCMAKE_TOOLCHAIN_FILE=../cmake/armM4.toolchain ..`
Beim Aufruf von `make` wird aus der pthreads-freien Vorgabe für ARM die Datei `stackanalyzer_arm` erzeugt, die nun weiter statisch analysiert werden kann.

☞ src/stackanalyzer_arm.c

6. Annotationen: Machen Sie sich mit den zur Verfügung gestellten Handbuch vertraut. Verschaffen Sie sich einen groben Überblick über die verfügbaren Annotationen.

7. Konfiguration: Starten Sie die Werkzeug-Suite mit folgendem Befehl:
`/proj/i4ezs/tools/a3_arm/bin/a3arm`.
Falls Sie nach einer Lizenzdatei gefragt werden, geben Sie folgenden Pfad an:
`/proj/i4ezs/tools/a3_arm/license.dat`
Machen Sie sich mit den verfügbaren Konfigurationsoptionen vertraut und wenden Sie diejenigen an, von denen Sie glauben, dass Sie notwendig sind oder Ihr Analyseergebnis positiv beeinflussen. *Welche sind das?*

.....
.....
.....

Beachten Sie insbesondere, dass `ais`-Annotationen im Quellcode standardmäßig nicht von der Analyse beachtet werden. Die kann durch entsprechende Konfigurati-

on aber geändert werden. Speichern Sie Ihr Projekt in Ihrem git-Repository ab und stellen Sie es unter Versionsverwaltung.

8. Analyse: Analysieren Sie nun den maximalen Stackverbrauch der `run()` Funktion und interessanter einzelner Module wie z.B. `find_maximum()`. *Woran scheitern die Analysen zunächst?*

.....

Wie können Sie den Werkzeugen helfen das Programm trotzdem zu analysieren? Notieren Sie sich die nötigen Anpassungen der Implementation oder Annotationen.

.....
.....
.....
.....

Vergleichen Sie die Ergebnisse für unterschiedliche Compileroptimierungen. Gerade `-O0` *und* `-Os` *sollten hier interessant sein.* Vergleichen Sie insbesondere den analysierten Stackverbrauch mit dem von Ihnen gemessenen. *Sind beide Werte aufgrund der unterschiedlichen Architekturen vergleichbar?*

.....

Wie verhält sich Ihre Messung im Verhältnis?

.....

Erweiterte Übung

Warteschlange

Diese Aufgabe kann wieder direkt auf der Intel-Architektur bearbeitet werden. Die statische Analyse auf der ARM Architektur ist optional. *Hinweis: Die Umstellung der ARM-Toolchain auf x86 erfolgt durch Löschen des Inhaltes des build Ordners.*

9. Vorbereitung: Kopieren Sie Ihre *Prioritätswarteschlange-Bibliothek aus der vorherigen Aufgabe* in die dafür vorgesehenen Dateien. Ändern Sie ihre Implementierung dahingehend ab, dass sich Prioriäten mehrfach einfügen lassen. Nutzen Sie Ihre Warteschlangenimplementierung aus der Datei `stackanalyzer_ext.c`, um die selben Daten wie in der grundlegenden Übung abzuspeichern. Iterieren Sie einmal über die ganze Warteschlange und suchen Sie auch hier nach dem größten Wert.

☞ `queue.h`
☞ `queue.c`
☞ `g_data`

.....

.....
.....
.....

Optionale Zusatzaufgabe: Statische Laufzeitanalyse

10. `aiT`**:** Uns steht nicht nur der `StackAnalyzer` sondern auch der `aiT` für die statische Bestimmung der Worst-Case-Execution-Time(WCET) zur Verfügung. In dieser optionalen Aufgabe, können Sie zusätzlich auch eine statische Laufzeitanalyse des Programms durchzuführen. Die Nutzung des `aiT` ist der des `StackAnalyzers` sehr ähnlich. Arbeiten Sie sich auf die gleiche Art und Weise in seine Benutzung ein.

Hinweise

- Bearbeitung: Gruppenarbeit
- Abgabezeit: 04.07.2016
- Fragen bitte an `i4ezs@lists.cs.fau.de`