

AUFGABE 6: STACKVERBRAUCHSANALYSE

In dieser Aufgabe werden Sie lernen, wie Sie nichtfunktionale Eigenschaften wie Stackverbrauch 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 eine einfache statische Analyse für eine Testanwendung durchführen um den maximalen Stackverbrauch auf eine sichere Art und Weise zu bestimmen. Abschließend werden Sie sich mit dem entsprechenden Werkzeug StackAnalyzer der Firma AbsInt vertraut machen.

1 Aufgabenstellung

Vermerken Sie Ihre Antworten zu den Fragen der einzelnen Aufgaben an den vorgesehenen Stellen in der vorgegebenen `answers.md`. Bitte erstellen Sie, um die Abgabe durch Mergerequests zu vereinfachen, **pro Aufgabe einen eigenen Branch**. Um einen konsistenten Zustand zu gewährleisten, benutzen Sie dazu bitte folgenden Befehl:
`git fetch git@gitlab.cs.fau.de:ezs/vezs22-vorgabe.git aufgabe6 &&
git checkout -b aufgabe6 FETCH_HEAD`

Wasserstand

Aufgabe 1 Vorgabe

Machen Sie sich mit unserer Vorgabe in `src/arrays.c`, `include/arrays.h` und `src/stackanalyzer.c` vertraut. Was wurde hier jeweils implementiert? Was leisten die Funktionen `fill()`, `find_max()`, `array_cpy()` und `sort()` sowie das Makro `GET_TOS()`? Wo liegt der Stapelspeicher für den Faden, der mit der Funktion `run()` gestartet wird?

Antwort:

Wir haben versucht, Ihnen so viel Ärger wie möglich mit der `pthread`-Bibliothek zu ersparen. Dennoch ist es eine gute Idee, sich mit den Handbuchseiten von `pthread_create` und `pthread_attr_setstack` auseinanderzusetzen, um die Aufrufe und ihre Parameter in der Datei `stackanalyzer.c` zu verstehen.

man 3p
pthread_create

man 3p
pthread_attr_setstack

Aufgabe 2 Messaufbau

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

```
❯ make
stackanalyzer_x86
```

```
❯ ./
stackanalyzer_x86
```

Aufgabe 3 Visualisierung

Um sich von der Korrektheit Ihrer Lösung zu überzeugen, ist es vorteilhaft, eine graphische Ausgabe des Stapelzustandes zu besitzen. In der Funktion `visualize()` finden sie eine Implementierungsskizze einer solchen Funktion, welche den Stapel dabei kompakt auf der Standardausgabe ausgibt. Dabei bezeichnet jedes ausgegebene Zeichen, wie viele Bytes des Wasserstandsmusters innerhalb von 32 Zeichen noch intakt geblieben sind. Machen Sie sich mit der Implementierung vertraut, erweitern Sie diese durch eine Erkennung Ihres Musters aus der vorherigen Aufgabe und rufen Sie die Visualisierung an geeigneter Stelle (oder auch Stellen) in Ihrem Programm auf. *Welche Regionen erkennen Sie in der Visualisierung? Erklären Sie insbesondere größere Blöcke, soweit möglich.*

Antwort:

Aufgabe 4 Messung

Messen Sie den Stack-Verbrauch der von uns vorgegebenen `run`-Funktion und der transitiv aufgerufenen Funktionen mit den von uns vorgegebenen Daten durch eine konkrete Ausführung des Programms.

Antwort:

```
❯ make
stackanalyzer_x86
```

```
❯ ./
stackanalyzer_x86
```

Überlegen Sie sich eigene Eingabewerte für die Datenstruktur, die den Speicherverbrauch maximieren. *Für welche Eingabesequenzen ergeben sich die höchsten Speicherverbräuche für die einzelnen Funktionen? Wie hoch ist dann der schlimmstmögliche Gesamtspeicherbedarf?*

Antwort:

Statische Stackanalyse

In dieser Teilaufgabe soll nun der Stapelverbrauch einzelner Funktionen durch statische Analyse bestimmt werden. Dazu wird das Flussproblem des maximalen Stapelverbrauchs wie in den Übungen gezeigt in ein mathematisches Optimierungsproblem umgewandelt und durch den Löser `lpsolve` ermittelt. Analysieren Sie hier den maximalen Stapelverbrauch der `run()` Funktion.

Aufgabe 5 Problemstellen

Verschaffen Sie sich zunächst einen Überblick über das Analyseproblem *und identifizieren Sie dort etwaige Problemstellen für die Analyse.*

Hinweis: Zur Anzeige des Analysegraphen im Target `make stackgraph_arrays` wird ferner eine laufende graphische Oberfläche benötigt.

Antwort:

```
❯ make
stackgraph_arrays
```

```
❯ remote.cip.
cs.fau.de,
Xpra
```

Wie können Sie diese Probleme durch Codeveränderung/Reimplementierung ausräumen oder in Ihrer Analyse berücksichtigen? Notieren Sie sich die nötigen Anpassungen der Implementierung.

Antwort:

Aufgabe 6 Analyse

Bestimmen Sie nun den maximalen Stapelverbrauch. Kodieren Sie dazu die relevanten Flussbedingungen als Nebenbedingungen für ein numerisches Maximierungsproblem für `lpsolve`. Als Ausgangspunkt können Sie die Datei `arrays.skel.lp` nutzen, welche automatisch im `build`-Verzeichnis erzeugt wird. Der Stapelbedarf kann nun mittels `./stackusage/lp_solvepp arrays.lp` ermittelt werden.

```
❯ make
stackgraph_arrays
```

Achtung: Die Datei `build/arrays.skel.lp` wird durch erneute Aufrufe von `make stackgraph_arrays` überschrieben, kopieren Sie sich deshalb die Datei und halten Sie diese Kopie nach dieser Änderung manuell aktuell. *Begründen Sie Ihren Lösungsweg, besonders bezüglich Konstanten in Ihrem Gleichungssystem.* Vergleichen Sie insbesondere den analysierten Stackverbrauch mit dem von Ihnen gemessenen. *Wie verhalten sich die Ergebnisse mit Bezug zu Ihrer Messung? Wie lassen sich etwaige Abweichungen erklären? Verbessern Sie bei Bedarf die Implementierung Ihrer Messung.*

Hinweis: Mit korrekten Eingaben sollte die Differenz zwischen statischem und dynamischem Wert nicht mehr als 200 Byte betragen. Greifen Sie bei Bedarf nochmals auf ihre Stapel-Visualisierung und den Aufrufgraphen zurück, um fehlerhafte Resultate zu identifizieren.

* GET_TOS

Antwort:

Statische Analyse mit der AbsInt-Werkzeugkette

Um einen Einblick in die statische Stapelverbrauchsanalyse „in der Praxis“ zu erhalten, können Sie in dieser Aufgabe die oben vorgestellte Anwendung auch mittels der in der Industrie verbreiteten Programme StackAnalyzer der Firma AbsInt untersuchen.

Aufgabe 7 Annotationen

Machen Sie sich mittels der zur Verfügung gestellten Folien sowie dem Handbuch mit der grundlegenden Bedienung des Programmes vertraut. Verschaffen Sie sich einen groben Überblick über die verfügbaren Annotationen.

Aufgabe 8 Konfiguration

Starten Sie die Werkzeug-Suite mit folgendem Befehl:

```
/proj/i4ezs/tools/a3_x86/bin/a3x86
```

Die Lizenz beziehen Sie dabei über unseren Lizenzserver. Nutzernamen und Passwort entsprechen dabei Ihren Anmeldedaten für den HelpDesk „Adora-Belle“, welche Sie am Anfang des Semesters erhalten haben. Kontaktieren Sie uns bitte, falls hierbei Probleme auftreten. Konfigurieren Sie die Analyse entsprechend der Vorgaben

* i4alm.cs.
fau.de
Port: 42424

aus den Folien (CPU-Variante, Binärdatei, Analysestart, Annotationsextraktion aus Quelltextdateien, etc.). Speichern Sie Ihr Projekt in Ihrem git-Repository ab und stellen Sie es unter Versionsverwaltung.

Hinweis: Die Warnungen der Analyse “ELF file is not an executable[...]” und “ELF file is not a statically linked executable[...]” sind durch die Natur der Anwendung (dynamisch gebundene Desktopanwendung) verursacht und dürfen durch Sie ignoriert werden. Alle anderen Warnungen/Fehler der Analyse sind jedoch durch Sie selbst geeignet zu adressieren.

Aufgabe 9 *Analyse*

Analysieren Sie nun den maximalen Stackverbrauch der Funktion `run()`. Optional können Sie gerne auch mit der Analyse weiterer Funktionen/Einstiegspunkte wie etwa `find_max()` experimentieren. *Woran scheitern die Analysen zunächst? Wie können Sie den Werkzeugen helfen, das Programm trotzdem zu analysieren? Notieren Sie sich die nötigen Anpassungen der Implementierung oder Annotationen. Wie verhalten sich diese Werte im Bezug zu Ihrer Messung, sowie Ihrer eigenen statischen Analyse?*

Antwort:

Hinweise

- Bearbeitung: Gruppenarbeit
- Abgabefrist: 27.01.2023
- Fragen bitte an i4ezs@lists.cs.fau.de