Grundlagenpraktikum Rechnerarchitektur (GRA)

Tutorübung

Moritz Beckel

09.05.2022 16:00 / 13.05.2022 15:00

Hausaufgaben

- Collatz
- EAN-13 Verifier
- Quiz

T3.1 Verwendung von Materialien

In dieser und den kommenden Wochen werden wir Ihnen für einige der Programmieraufgaben Materialvorlagen bereitstellen, die Ihnen das bearbeiten der Aufgaben erleichtern. Diese enthalten üblicherweise ein Rahmenprogramm, welches grundlegende I/O-Funktionalitäten bereitstellt, und ein `Makefile`, welches Anweisungen zum Kompilieren des Programms enthält.

1. Laden Sie die Vorlage herunter:

   ```
   wget https://gra.caps.in.tum.de/m/gcd.tar
   ```

2. Extrahieren Sie die Dateien aus dem Tar-Archiv:

   ```
   tar xf gcd.tar
   ```

3. Wechseln Sie in den neu angelegten Ordner `gcd`: `cd gcd`

4. Führen Sie den Befehl `ls` aus und vergewissern Sie sich, dass sich in diesem Ordner die Dateien `gcd.c` und `Makefile` befinden.

T3.1 Verwendung von Materialien

5. Öffnen Sie nun die Datei `gcd.c` mit einem Texteditor Ihrer Wahl.

   Diskutieren Sie mit Ihrem Tutor die Bedeutung folgender Zeilen:

   - `#include <stdio.h>`
   - `void test(int, int);`
   - `int main(int argc, char** argv)`
   - `printf(...);` – Nutzen Sie hierzu auch die man-Page `man 3 printf`.

T3.1 Verwendung von Materialien

6. Kompilieren Sie das Programm mittels `make` und führen Sie es aus. Verhält sich das Programm wie erwartet?

7. Bevor wir das Programm debuggen werden, lassen Sie sich die Disassembly des Programms anzeigen (`objdump -d gcd | less`). Können Sie die rekursiven Funktionsaufrufe identifizieren?

8. Öffnen Sie die Datei `Makefile`[1], passen Sie das Compiler-Flag `-O0` zu `-O2` an (was verändert dies?) und speichern Sie Datei. Löschen Sie die alte Binary mit `make clean` und kompilieren Sie das Programm erneut mit `make`. Wie verhält sich das Programm nun?

T3.1 Verwendung von Materialien

9. Lassen Sie sich erneut die Disassembly des Programms anzeigen (`objdump -d gcd | less`). Können Sie die rekursiven Funktionsaufrufe identifizieren?

10. Ändern Sie das Compiler-Flags im Makefile erneut zu `-O0` und fügen Sie zusätzlich das Flag `-g`^{2} hinzu. Kompilieren Sie das Programm erneut (`make clean gcd`).

11. Starten Sie den Debugger GDB (`gdb ./gcd`) und starten Sie das Programm (`r`). Lassen Sie sich den Backtrace anzeigen (`bt`). Erkennen Sie das Problem?

    Hinweis: Sie können auch ein ausgeführtes GDB-Kommando jederzeit mit *Ctrl-C* abbrechen.

12. Überlegen Sie, wie Sie das Problem lösen könnten.

Hausaufgaben

P3.1 Summe einer Linked List [2 Pkt.]

Eine *Linked List* ist eine Datenstruktur, wo jedes Element der Liste zusätzlich einen Pointer auf das nachfolgende Element enthält; oder `NULL`, falls es kein nachfolgendes Element gibt. Die leere Liste ist daher einfach ein `NULL`-Pointer. Schreiben Sie in C eine Funktion, die für eine Linked List mit Elementen vom Datentyp `uint64_t` die Summe der enthaltenen Elemente berechnet:

```
struct node { struct node* next; uint64_t val; };
uint64_t listsum(const struct node* list);
```

Hausaufgaben

P3.2 Brainfuck-Interpreter [4 Pkt.]

Brainfuck[4] ist eine esoterische, aber dennoch Turing-vollständige Programmiersprache. Als Zustand gibt es ein (theoretisch) unendlich großes Byte-Array und einen verschiebbaren Pointer auf diesen. Sie besteht aus lediglich acht Befehlen (alle anderen Zeichen, mit Ausnahme des Null-Bytes zum Anzeigen des Programmendes, werden als Kommentare behandelt und ignoriert):