Programmieren in C

SS 2019

Vorlesung 12, Dienstag 23. Juli 2019

(Debugger, Sanitizer, +X)

Prof. Dr. Peter Thiemann
Professur für Programmiersprachen
Institut für Informatik
Universität Freiburg

Auf der Grundlage eines Foliensatzes von Prof. Dr. Bast

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem Projekt — 4 gewinnt
  - Ankündigungen
- Inhalt
  - Debugging — gdb
  - Debugging — AddressSanitizer
  - C Programmierung (falls Zeit) — Funktionszeiger

Erfahrungen mit dem Projekt 1/2

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Schon einige (wenige) vollständige Abgaben
  - Bisher positive Erfahrungen (negative kommen meist erst später)

Erfahrungen 2/2

UNI FREIBURG

- Aus der Evaluation
  - Programmieren in C im SS 2019
  - Aufwand: 64% sehr hoch, 16% hoch, 17% angemessen, 2% gering, 2% sehr gering
  - Informatik Durchschnitt
  - Aufwand: 23% sehr hoch, 27% hoch, 46% angemessen, 3% gering, 1% sehr gering
- Analyse
  - Für 6 ECTS (entsprechend 180h) sind 10h/Woche erforderlich
  - Durchschnitt Erfahrungen: max 10h mit großer Varianz
  - Alternative: Kurs geht über die Vorlesungszeit hinaus

Debugging 1/4

UNI FREIBURG

- Fehler im Programm kommen vor (wdhl)
  - Mit Feldern, Zeigern und malloc() lassen sich unangenehme **seg**mentation **faults** produzieren
  - Das passiert beim versuchten Zugriff auf Speicher, der dem Programm nicht gehört, zum Beispiel

```
int* p = NULL;    // Pointer to address 0.
*p = 42;          // Will produce a segmentation fault.
```

  - Schwer zu debuggen, es kommt dann einfach etwas wie:

```
Segmentation fault (core dumped)
```

    Ohne Hinweis auf die Fehlerstelle im Code (gemein)
  - Manche Fehler sind zudem nicht deterministisch, weil sie von nicht-initialisiertem Speicherinhalt abhängen

Debugging 2/4

UNI FREIBURG

- Methode 2: gdb, der **GNU debugger**
  - Gbd Features
    - Anweisung für Anweisung durch das Programm gehen
    - Sogenannte breakpoints im Programm setzen und zum nächsten breakpoint springen
    - Werte von Variablen ausgeben (und ändern)
  - **Vorteil:** beschleunigte Fehlersuche im Vgl zu printf
  - **Nachteil:** ein paar gdb Kommandos merken

Debugging 3/4

UNI FREIBURG

- Grundlegende gdb Kommandos
  - **Wichtig:** Programm kompilieren mit der –g Option!
  - gdb aufrufen, z.B. gdb ./ArraysAndPointersMain
  - Programm starten mit run <command line arguments>
  - stack trace (nach seg fault) mit backtrace oder bt
  - breakpoint setzen, z.B. break Number.cpp:47
  - breakpoints löschen mit delete oder d
  - Weiterlaufen lassen mit continue oder c
  - Wert einer Variablen ausgeben, z.B. print x oder p i

Debugging 4/4

UNI FREIBURG

- Weitere gdb Kommandos
  - Nächste Zeile im Code ausführen step bzw. next

    step folgt Funktionsaufrufen, next führt sie ganz aus
  - Aktuelle Funktion bis zum return ausführen finish
  - Aus dem gdb heraus make ausführen make
  - Kommandoübersicht / Hilfe help oder help all
  - gdb verlassen mit quit oder q
  - Wie in der bash command history mit Pfeil hoch / runter

    Es geht auch Strg+L zum Löschen des Bildschirmes

AddressSanitizer 1/6

UNI FREIBURG

- AddressSanitzer: Auffinden von Speicherfehlern
  - Use after free (dangling pointer dereference)
  - Heap buffer overflow
  - Stack buffer overflow
  - Global buffer overflow
  - Memory leaks
- Zur Verwendung
  - Compilieren und Linken mit Flag –g -fsanitize=address
  - Weitere Steuerung durch Umgebungsvariable ASAN_OPTIONS

AddressSanitizer 2/6

UNI FREIBURG

- Use after free (Verwendung nach Aufruf von free)

```
#include <malloc.h>
int main(int argc, char * argv[]) {
  char * buffer = malloc(10);
  if (buffer) {
    buffer[0] = 'x';
    free(buffer);
    buffer[0] = 'y';            /* use after free */
  }
  return 0;
}
```

AddressSanitizer 3/6

UNI FREIBURG

- Heap Buffer Overflow

```
#include <malloc.h>

int main(int argc, char * argv[]) {
  char * buffer = malloc(10);
  if (buffer) {
    buffer[10] = 'x';            /* heap buffer overflow */
  }
  return 0;
}
```

AddressSanitizer 4/6

UNI FREIBURG

- Stack Buffer Overflow

```
int main(int argc, char * argv[]) {
  char buffer[10];
  buffer[10] = 'x';            /* stack buffer overflow */
  return 0;
}
```

AddressSanitizer 5/6

UNI FREIBURG

- Global Buffer Overflow

```
char buffer[10] = {0};

int main(int argc, char * argv[]) {
  buffer[10] = 'x';            /* global buffer overflow */
  return 0;
}
```

AddressSanitizer 6/6

UNI FREIBURG

- Memory Leaks

```
#include <malloc.h>
int main(int argc, char * argv[]) {
  char * buffer = malloc(10);
  if (buffer) {
    buffer[0] = 'x';
  }
  return 0;                    /* leaks buffer */
}
```

Literatur / Links

- Debugger / gdb
  - https://www.gnu.org/software/gdb/
  - https://www.cs.cmu.edu/~gilpin/tutorial/
  - https://www.geeksforgeeks.org/gdb-step-by-step-introduction/
- Debugger / AddressSanitizer
  - https://github.com/google/sanitizers/wiki/AddressSanitizer
- Funktionszeiger
  - https://www.learn-c.org/en/Function_Pointers