Programmieren in C

SS 2021

Vorlesung 12, Dienstag 13. Juli 2021

(zum Projekt, Vermischtes)

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

Auf der Grundlage eines Foliensatzes von Prof. Dr. Bast

Heutige Vorlesung

UNI FREIBURG

- Organisatorisches
  - Evaluation
  - Vorstellung des Projekts — Mine Sweeper
- Inhalt
  - Debugging

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

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