# Debugging mit GDB

Proseminar - „C - Grundlagen und Konzepte“

22. Mai 2014 - Albrecht Oster

# Gliederung

- Was ist Debugging?
- Was ist GDB?
- Fallbeispiel eines Problems
- Wie kann uns GDB an dieser Stelle helfen?
- Weitere Einsatzmöglichkeiten

# Was ist Debugging?

- Software auf Fehler untersuchen
  - Testing
  - Produktiveinsatz
- Fehlerbehebung
- Kann zeitaufwendig und mühsam sein

# Was ist GDB?

- GDB = GNU Debugger
- Auf eine Vielzahl an Sprachen, Plattformen und Architekturen portiert
- Kommandozeilenprogramm
  - In viele Entwicklungsumgebungen grafisch integriert
- Erleichtert die Ursachenforschung eines Fehlers zur Laufzeit

# Fallbeispiel

```c
#include <stdio.h>

void schreibeZahlen(int zahlen[], int anzahl)
{
  for (int i = 0; i < anzahl; i--)
  {
    zahlen[i] = i + 1;
  }
}

void gibZahlAus(int zahlen[], int index)
{
  printf("%d\n", zahlen[index]);
}
```

```c
void gibZahlenAus(int zahlen[], int anzahl)
{
  for (int i = anzahl; i >= 0; i++)
  {
    gibZahlAus(zahlen, i);
  }
}

int main(int argc, char *argv[])
{
  int zahlen[10];
  schreibeZahlen(zahlen, 10);
  gibZahlenAus(zahlen, 10);
  exit(0);
}
```

# Fallbeispiel

***Demo***

Ergebnis der Ausführung des kompilierten Codes

Programm stürzt ab

# Was können wir jetzt tun?

- Code hilfsweise bearbeiten
  - Testen, bis zu welcher Stelle der Code ausgeführt wird (mit **`printf`**)
  - Zwischenergebnisse ausgeben
  - hilfreich bei Fehlern unter besonderen Bedingungen

```
11. void gibZahlAus(int zahlen[], int index)
12. {
13.   printf("Zahl an Index %d:“, index);
14.   printf("%d\n", zahlen[index]);
15. }

23. int main(int argc, char *argv[])
24. {
25.   int zahlen[10];
26.   schreibeZahlen(zahlen, 10);
27.   printf("Zahlen in Array geschrieben.\n");
28.   gibZahlenAus(zahlen, 10);
29.   gibZahlenAus(zahlen, 10);
30.   exit(0);
31. }
```

# Wie kann uns GDB an dieser Stelle helfen?

- GDB „klinkt“ sich in den internen Ablauf des Programms ein
  - Kompilieren: `gcc example.c -g -o example`
  - Ausführen: `gdb example`
- Pausieren des Programms an beliebiger Stelle durch „Breakpoints“
  - `break [`*`Datei`*`].c:[`*`Zeile`*`]` oder `break` *`my_func`*
- Einblick in RAM: Variablenwerte zur Laufzeit einsehen
  - `print` *`my_var`*, `watch` *`my_var`*

# Die wichtigsten Befehle

- Zeilenweises Durchlaufen des Codes
  - `step`, `next`, `continue`
- Stackframes untersuchen
  - `frame, up, down`
  - `backtrace`

# Wie kann uns GDB an dieser Stelle helfen?

***Demo***

Ausführen des Programms über gdb, durchlaufen, verschiedene Befehle vorführen, „Absturzstelle“ finden

# GDB in Entwicklungsumgebungen

- GDB ist mit einer GUI sehr intuitiv zu bedienen
- Integriert in: *Eclipse*, *Anjuta*, *Netbeans*, *Xcode*, *QT Creator*
- Zusammenarbeit mit anderen Tools
  - *valgrind*

# GDB in Entwicklungsumgebungen

***Demo***

IDE mit unserem Code aufrufen und GDB diesmal per GUI steuern
Stackframes erläutern und vorführen

# Literatur

- Wikipedia: „GNU Debugger“
  https://en.wikipedia.org/wiki/GNU_Debugger
- GDB: The GNU Project Debugger
  http://www.gnu.org/software/gdb/documentation/
- Samuel Huang - GDB Tutorial
  http://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf
- O’Reilly - Debuggen mit gdb
  http://www.oreilly.de/german/freebooks/rlinux3ger/ch142.html