GDB

Starten, Allgemeines

Vorraussetzung: Debug Symbole => Programm mit '-g' oder '-ggdb' kompilieren

Level 0 bis 3 möglich => -ggdb3 oder -g1, default ist Level 2 [1]

Vorsicht bei IDEs die beim linken entweder '-s' oder beim installieren 'strip' verwenden

=> debug Symbole verschwinden

GDB starten:

```
gdb <path-to-program>
```

Logging (optional):

```
(gdb) set logging file backtrace.log
(gdb) set logging on
```

man kann jederzeit 'set logging off' setzen

Programm starten:

```
(gdb) run [program arguments]
```

Segmentation Fault debuggen

```
(gdb) backtrace
#0 0x0000000000401cc8 in funktion_1 (arg=0x604050)
   at main.c:320
#1 0x0000000000401233 in funktion_c (arg=0x674430)
   at main.c:129
#2 0x0000000000400d3c in main ()
   at main.c:22
(gdb) frame 0
#0 0x0000000000401cc8 in funktion_1 (arg=0x604050)
320       printf("%d", node → task → start);
(gdb) list
...Ausgabe vom code-block...
(gdb) info local
...Ausgabe aller lokalen Variablen...
(gdb) print node
$1 = (task_node *) 0x0
```

hier ist unser Fehler: ein **Null-Pointer** wird dereferenziert!

Allgemeines Debugging/Beobachtung

```
(gdb) break main.c:311
(gdb) break main.c:del_node
(gdb) watch to_do_list
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x40f37 in add_node at main.c:311
2       watchpoint     keep y                      to_do_list
3       breakpoint     keep y   0x40b92 in del_node at main.c:99
(gdb) run
Breakpoint 3, del_node () at main.c:99
99        newnode = mynode;
(gdb) next
100       printf("%s", newnode → task → description);
print  newnode → task → description
$1 = "Praktikum verteilte Systeme", '\000' <repeats 12 times>
(gdb) continue
Breakpoint 1, add_node () at main.c:311
311       if (time_overlap(node1, node2)) {
(gdb) step
time_overlap (node1=0x4025b9, node2=0x4038f1) at main.c:681
681       if (node1 == NULL || node2 == NULL) {
(gdb) finish
...beendet gesamte Funktion und zeigt uns den return-Wert...
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

beachten bei Watchpoints:

- watchpoints auf lokale Variablen können erst gesetzt werden, wenn wir in der entsprechenden Funktion sind (beim Verlassen wird der watchpoint gelöscht)
- watchpoints auf globale Variablen werden über die gesamte Ausführung beobachtet
- bei jeder Änderung der beobachteten Variable wird die Ausführung unterbrochen

Cheat sheet

help [command]	allgemeine oder spezifische Hilfe zu einem gdb-Kommando
Start und Stop	
run [args]	startet das Programm, optional mit Argumenten
kill	beendet die momentane Ausführung des Programms
quit	beendet GDB
Var-/Codeinfo	
print <var>	gibt den Wert eine Variablen aus (z.b. Addresse eines Pointers)
list [Zeile]	gibt den Code-Block aus in dem wir sind oder den von [Zeile]
Stack	
backtrace	gibt den call stack aus (Liste der nacheinander ausgeführten Funktionen)
frame [num]	gibt einen spezifischen Funktionsaufruf im call stack aus und die codezeile
info frame [num]	mehr informationen über einen frame/Funktionsaufruf
info locals	zeigt alle lokalen Variablen, die gerade verfügbar sind
info args	zeigt die Argumente an, mit denen die Funktion aufgerufen wurde
Break-/Watchpoints	
break [file:]<Zeile/func>	setzt Breakpoint in file, bei angegebener Zeile/Funktion
watch <var>	beobachtet alle Änderungen an einer Variablen
info breakpoints	zeigt alle Breakpoints (und Watchpoints) an
info watchpoints	zeigt alle Watchpoints an
disable <num>	Deaktiviert einen Watchpoint/Breakpoint
ignore <num> <x>	ignoriert einen Wachtpoint/Breakpoint x mal
Navigation	
next [num]	geht zur nächsten Zeile (next 5 => 5 Zeilen weiter)
step [num]	geht zur nächsten Zeile und springt in den nächsten Funktionsaufruf (wenn vorhanden), step 2 => springe max 2mal
until <Zeile>	führe das Programm weiter bis zu <Zeile> aus
continue	führe das Programm weiter aus bis zum nächsten break-/watchpoint
finish	beende die momentane Funktion vollständig
Aufrufen/Zuweisen	
set <var> = <value>	setze einen neuen Wert für eine Variable
call <func>	rufe manuell eine Funktion aus dem Programm auf

LINKS

[1] gcc: Debugging Options

Valgrind Quick Start guide (alternatives tool)

GUIs

Data Display Debugger

Nemiver

KDbg

Insight

Tutorials

RMS's gdb Debugger Tutorial

O'REILLY Debuggen mit gdb