1 GDB Tutorial

Willkommen in der Computergrafik-Übung!

Dieses Dokument soll Sie in den Umgang mit dem GNU Debugger (kurz GDB) einführen. Diesen können Sie z.B. in der virtuellen Maschine einsetzen, um das Laufzeitverhalten Ihres Programmes nachzuvollziehen. Wenn Sie hingegen mit Visual Studio oder einer anderen IDE arbeiten, so empfiehlt es sich den dort integrierten Debugger einzusetzen. Die grundlegende Funktionalität der verschiedenen Debugger ist jedoch sehr ähnlich.

1.1 Einrichtung in der VM

In der virtuellen Maschine ist GDB nicht vorinstalliert. Dies können Sie aber mit einem einfachen Befehl nachholen. Geben Sie dafür in ein Konsolenfenster den Befehl „sudo pacman -S gdb“ ein. Sollten hierbei Fehler wie „failed retrieving file“ auftreten, so müssen Sie vorher die Paketquellen aktualisieren. Dies können Sie mit dem Befehl „sudo pacman -Syu“ ausführen.

Hinweis: Die VM benötigt dafür Internetzugriff.

1.2 Programmcode mit Debug-Symbolen erstellen

Damit GDB in der Lage ist, Ihnen den Programmfluss anhand des Quelltextes zu veranschaulichen, müssen Sie das Programm mit den sogenannten Debug-Symbolen erstellen. Wenn Sie eine IDE verwenden, sollten Sie dies als als Build-Target auswählen können. Im Folgenden wird erklärt, wie Sie dies mit CMake über die Konsole einrichten können.

Wenn Sie das Projekt mit CMake und Make bauen, so müssen Sie CMake explizit angeben, dass das Projekt mit Debug-Symbolen erstellt werden soll. Dies ist über den Kommandozeilenparameter „-DCMAKE_BUILD_TYPE=Debug“ möglich. Es empfiehlt sich, für den Debug-Build ein zweites Verzeichnis anzulegen. Analog zu der in der Übungsaufgaben-Anleitung beschrieben Methode das Build-Verzeichnis mit „cmake ../“ können Sie nun mit

make –DCMAKE_BUILD_TYPE=Debug ../

das Build-Verzeichnis für Ihren Debug-Build erstellen. Dies ist ebenfalls nur einmal pro Aufgabe nötig.

1.3 GDB Funktionsprinzip

GDB führt ein Programm grundsätzlich normal aus. Sie können jedoch sowohl vor Programmstart, als auch während der Ausführung Haltepunkte, sog. Breakpoints setzen. Damit legen Sie Stellen im Programm fest, an welchen die Ausführung unterbrochen werden soll. Während dieser Unterbrechung können Sie die Variablen des aktuellen Stackframes einsehen, aber auch den gesamtem Ausführungsstack betrachten. Ebenfalls können Sie das Programm Zeile für Zeile einzeln ausführen und somit besser dem Programmverlauf folgen. So können sie auch feststellen, wo genau ein Fehler auftritt. GDB wird das Programm selbstständig unterbrechen, wenn es zu einem kritischen Fehler in der Ausführung kommt.

1.4 GDB Beispiele

Die folgenden Minimalbeispiele sollen die grundlegende Bedienung von GDB veranschaulichen.

1.4.1 Beispiel 1

```
1 int main() {
2         int * a = nullptr;
3         int b = *a;
4         return 0;
5 }
```

Wenn versucht wird, dieses Programm direkt auszuführen, wird als Fehler nur ein „Segmentation fault“ ohne genauere Angabe ausgegeben.

Nun soll das Programm mit GDB ausgeführt werden, Hierfür wird GDB auf der Kommandozeile mit dem entsprechenden Programm als Parameter aufgerufen. Der Programmname lautet hierbei „nullptr“.

```
gdb ./nullptr
```

GDB wird gestartet und lädt das Programm. Allerdings wird das Programm noch nicht ausgeführt. An dieser Stelle ist es nun möglich, Breakpoints zu setzen. Dies wird im zweiten Beispiel verdeutlicht.

Mit dem Befehl „run“ kann das Programm nun gestartet werden. Erwartungsgemäß wird nun auch hier das Programm mit einem „Segmentation fault“ abstürzen. Wenn das Programm jedoch richtig mit Debug-Symbolen gebaut wurde, dann gibt GDB nun eine Fehlermeldung wie diese aus:

```
(gdb) run
Starting program: /home/cg/nullptr

Program received signal SIGSEGV, Segmentation fault.
0x0804847b in main () at nullptr.cpp:3
3                 int b = *a;
```

Wie zu erkennen ist, trat der Fehler in Zeile 3 auf. Mit dem Befehl „print“ kann nun auch genauer untersucht werden, welchen Wert die Variablen haben.

```
(gdb) print a
$1 = (int *) 0x0
(gdb) print b
$2 = 1
```

1.4.2 Beispiel 2

```
1  int main() {
2          int a = 10;
3          int b = 0;
4
5          b += 1;
6          b /= 2;
7
8          int c = a / b;
9
10         return 0;
11 }
```

Im Folgenden soll in dem Beispielprogramm genauer untersucht werden, welche Werte die Variablen annehmen. Dafür wird wieder GDB gestartet. Diesmal wird jedoch ein Breakpoint in Zeile 3 gesetzt und anschließend das Programm ausgeführt.

```
[cg@mybox ~]$ gdb ./divzero
GNU gdb (GDB) 7.10
...
Reading symbols from ./divzero...done.
(gdb) break divzero.cpp:3
Breakpoint 1 at 0x8048478: file divzero.cpp, line 3.
(gdb) run
Starting program: /home/cg/divzero

Breakpoint 1, main () at divzero.cpp:3
3                int b = 0;
(gdb) print a
$1 = 10
(gdb) print b
$2 = 1
```

Der Breakpoint wird dabei mit dem Befehl „break Datei:Zeile“ gesetzt. Wie in der Ausgabe zu sehen ist, hat GDB die Ausführung unterbrochen, als Breakpoint 1 erreicht wurde. Bei den Breakpoints ist zu beachten, dass diese **vor** der Zeile stehen, also der Inhalt der Zeile selbst noch nicht ausgeführt wurde. Dies hat in diesem Beispiel den Nebeneffekt, dass der Variable *b* noch keinen Wert zugewiesen wurde. Sie hat daher zu diesem Zeitpunkt einen undefinierten Wert, der rein zufällig 1 ist.

Mit dem Befehl „next“ kann nun eine Programmzeile ausgeführt werden. In diesem Fall wird nun Zeile 3 ausgeführt und zu der nächsten Codezeile gesprungen.

```
(gdb) next
5                b += 1;
(gdb) print a
$3 = 10
(gdb) print b
$4 = 0
```

Mit dem Befehl „watch“ können Variablen beobachtet werden, wodurch GDB eine Änderung der Variable mit vorherigem und neuem Wert in die Konsole schreibt.

```
(gdb) watch b
Hardware watchpoint 2: b
(gdb) next
Hardware watchpoint 2: b

Old value = 0
New value = 1
main () at divzero.cpp:6
6                 b /= 2;
(gdb) next
Hardware watchpoint 2: b

Old value = 1
New value = 0
main () at divzero.cpp:8
8                 int c = a / b;
```

Wie hier zu beobachten ist, hat die Ganzzahldivision der Variable *b* in Zeile 6 eine Null ergeben, weshalb es in der folgenden Zeile zu einer Division durch Null kommt.

```
(gdb) next

Program received signal SIGFPE, Arithmetic exception.
0x08048496 in main () at divzero.cpp:8
8                 int c = a / b;
```

1.5 Liste wichtiger GDB Befehle

Hier sind einige Befehle und deren Abkürzungen aufgelistet. Diese Liste ist jedoch keinesfalls vollständig und soll nur als schneller Überblick dienen.

Befehl	Abkürzung	Bedeutung
run	r	startet das Programm
break <Datei:Zeile>	b	setzt einen Breakpoint in die Zeile
break <Datei:Funktion>	b	setzt einen Breakpoint auf die Funktion
watch <Variable>	w	beobachtet eine Variable auf Veränderungen
print <Variable/Ausdruck>	p	gibt den Wert aus, bzw. wertet den Ausdruck aus
next	n	führt eine weitere Zeile aus
step	s	wie next, folgt jedoch Funktionsaufrufen
continue	c	setzt die Ausführung fort
info breakpoints	-	listet alle Breakpoints auf
delete	d	entfernt alle Breakpoints, Watchpoints und Catchpoints
where	-	zeigt die aktuelle Position an
quit	q	beendet GDB