Debuggen mit gdb

sECuRE beim NoName e.V.

powered by LaTeX, of course

26. Februar 2009

Inhalt

Bevor es losgeht (1)

- Variadic macros \o/

```
#define DEBUG(message, ...) \
printf("%s:%d:" message, __FILE__, __LINE__, __VA_ARGS__);

DEBUG("TIZ INFORMASHUN CUD BE RELEVANT TO YAR INTERESTS\n");
DEBUG("x = %d\n", x);
```

- assert (Zusicherungen), die man während dem Testen benutzt

```
#include <assert.h>
void foo(int *bar) {
        assert(bar != NULL);
        *bar = 23;
}
int main() { foo(NULL); }
```

```
assertion "bar != NULL" failed: file "test.c", line 5,
function "foo"
```

Bevor es losgeht (2)

- valgrind, findet Memory leaks (zuverlässig) und Fehler beim Speicherzugriff (nicht ganz so gut)

```
$ valgrind -v --log-file=vg --num-callers=20 \
  --leak-check=full ./programm
$ less vg
```

- loggt nach vg, bei threads gibt's vg.<pid>
- gibt einen 20 Zeilen langen Callstack an
- führt einen vollen Speichercheck durch
- ⇒ alle Fehler beheben, bevor man weiterhin debugged

Wie man korrekt kompiliert

- Debugging symbols standardmäßig vorhanden, außer bei gcc -s
- -g erzeugt debug information, zusätzlich -O0 am besten
- -g3 erzeugt mehr debug information, -gdwarf-2 beinhaltet auch Makros
- Am besten ein simples Makefile schreiben:

```
CFLAGS += -gdwarf-2
CFLAGS += -g3

all: foo
```

Dann:

```
$ make
cc -gdwarf-2 -g3   -c -o foo.o foo.c
```

Grundwissen gdb

- list [foo.c:13] - Zeigt sourcecode an
- info functions - alle Funktionen mit symbols
- run - startet das Programm
- ˆC - unterbricht das Programm
- print <variable> - zeigt den Wert einer Variablen an
- info locals - zeigt alle lokalen Variablen
- break [foo.c:13] - setzt einen Breakpoint
- <return> - führt den letzten Befehl nochmal aus

Wenn es knallt

```
$ ./core
variable is 23
zsh: segmentation fault (core dumped)  ./core
$ gdb core core.core
[...]
Core was generated by ‘core’.
Program terminated with signal 11, Segmentation fault.
#0  0x0000000000400920 in print_int (variable=0x0) at core.c:4
4               printf("variable is %d\n", *variable);
(gdb)
```

```
(gdb) backtrace
#0  0x0000000000400920 in print_int (variable=0x0) at core.c:4
#1  0x0000000000400960 in do_stuff (b=23) at core.c:11
#2  0x0000000000400970 in main () at core.c:15
```

```
(gdb) frame 1
#1  0x0000000000400960 in do_stuff (b=23) at core.c:11
11              print_int(a);
```

Breakpoints

```
char *bleh = "Yip Yip";
char foo[5] = "0000";
void do_stuff(int position) {
        printf("do_stuff()\n");
        printf("martians say: %s\n", bleh);
        foo[position] = 'x';
        printf("foo changed to: %s\n\n", foo);
}
[...]
```

```
$ ./break
do_stuff()
martians say: Yip Yip
foo changed to: x000

do_stuff()
martians say: Yip Yip
foo changed to: x000

do_stuff()
zsh: bus error (core dumped)  ./break
```

Breakpoints (2)

```
$ gdb break break.core
[...]
Core was generated by ‘break’.
Program terminated with signal 10, Bus error.
#0  0x00007f7ffdbcaf9a in strlen () from /usr/lib/libc.so.12
(gdb) backtrace
#0  0x00007f7ffdbcaf9a in strlen () from /usr/lib/libc.so.12
#1  0x00007f7ffdbc3c4d in __vfprintf_unlocked () from /usr/lib/libc
#2  0x00007f7ffdbc4d64 in vfprintf () from /usr/lib/libc.so.12
#3  0x00007f7ffdbc005e in printf () from /usr/lib/libc.so.12
#4  0x000000000040098b in do_stuff (position=1) at break.c:8
#5  0x00000000004009cf in main () at break.c:16
(gdb)
```

Warum stürzt es ab?!

Breakpoints (3)

```
$ gdb break
(gdb) list
6       void do_stuff(int position) {
7               printf("do_stuff()\n");
8               printf("martians say: %s\n", bleh);
9               foo[position] = 'x';
10              printf("foo changed to: %s\n\n", foo);
11      }
(gdb) break 7
Breakpoint 1 at 0x40096b: file break.c, line 7.
(gdb) r
Starting program: /home/michael/NoName/gdb/demo/break/break

Breakpoint 1, do_stuff (position=0) at break.c:7
7               printf("do_stuff()\n");
(gdb) next
do_stuff()
8               printf("martians say: %s\n", bleh);
```

Breakpoints (4)

```
(gdb) print bleh
$1 = 0x400a71 "Yip Yip"
(gdb) next
martians say: Yip Yip
9               foo[position] = 'x';
(gdb) continue
Continuing.

Breakpoint 1, do_stuff (position=-1) at break.c:7
7               printf("do_stuff()\n");
(gdb) next
do_stuff()
8               printf("martians say: %s\n", bleh);
(gdb) next
martians say: Yip Yip
9               foo[position] = 'x';
(gdb) next
10              printf("foo changed to: %s\n\n", foo);
(gdb) print bleh
$2 = 0x7800000000400a71 <Address 0x7800000000400a71 out of bounds>
```

Conditional breakpoints

```
$ gdb break
(gdb) list
6       void do_stuff(int position) {
7               printf("do_stuff()\n");
8               printf("martians say: %s\n", bleh);
9               foo[position] = 'x';
10              printf("foo changed to: %s\n\n", foo);
11      }
(gdb) break 7 if position < 0
Breakpoint 1 at 0x40096b: file break.c, line 7.
(gdb) run
Starting program: /home/michael/NoName/gdb/demo/break/break
do_stuff()
martians say: Yip Yip
foo changed to: x000



Breakpoint 1, do_stuff (position=-1) at break.c:7
7               printf("do_stuff()\n");
```

Watchpoints

```
$ gdb break
(gdb) watch bleh
Watchpoint 1: bleh
(gdb) run
Starting program: /home/michael/NoName/gdb/demo/break/break
Watchpoint 1: bleh
Watchpoint 1: bleh
do_stuff()
martians say: Yip Yip
foo changed to: x000

do_stuff()
martians say: Yip Yip
Watchpoint 1: bleh

Old value = 0x400a71 "Yip Yip"
New value = 0x780000000400a71 <Address 0x780000000400a71 out of b
do_stuff (position=-1) at break.c:10
10              printf("foo changed to: %s\n\n", foo);
(gdb)
```

Abkürzungen

- run = r
- break = b
- next = n
- continue = c
- print = p
- list = l
- frame = f
- backtrace = bt

Beispiel

```
Core was generated by ‘/usr/bin/i3’.
Program terminated with signal 6, Aborted.
[New process 12721]
#0  0x00007f0e0c484ed5 in raise () from /lib/libc.so.6
(gdb) bt
#0  0x00007f0e0c484ed5 in raise () from /lib/libc.so.6
#1  0x00007f0e0c4863f3 in abort () from /lib/libc.so.6
#2  0x00007f0e0c4c13a8 in ?? () from /lib/libc.so.6
#3  0x00007f0e0c4c6948 in ?? () from /lib/libc.so.6
#4  0x000000000040a7ff in initialize_xinerama (conn=0x6112a0) at sr
#5  0x0000000000408ba3 in main (argc=1, argv=0x7fff150d1cf8, env=0x
(gdb) frame 4
#4  0x000000000040a7ff in initialize_xinerama (conn=0x6112a0) at sr
135             free(screen_info);
```

Beispiel (2)

```
(gdb) list
130                     memcpy(&(workspaces[num_screens++].rect), &
131                     printf("that is virtual screen at %d x %d w
132
s->rect.x, s->rect.y, s->rect.width, s->rect.height);
133             }
134
135             free(screen_info);
136     }
(gdb) p screen_info
$1 = (xcb_xinerama_screen_info_t *) 0x61b090
(gdb) p screen_info->width
$2 = 1280
(gdb) p screen_info->height
$3 = 800
(gdb)
```

Macros

- noch nicht komplett implementiert (gdb 6.5)
- macro list gibt es noch nicht
- stringification (#x) wird noch nicht expanded
- beim Debuggen sind macro-definitionen scope-gebunden

Macros (Beispiel)

```
$ gdb macros
(gdb) list
3       #define I_HAS_A_MACRO(x) printf("the value of " #x " is %s\
4
10      int main() {
11              #define BLEH(bleh) bleh
12              do_sumfing();
(gdb) b 7
Breakpoint 1 at 0x400914: file macros.c, line 7.
(gdb) r
Starting program: /home/michael/NoName/gdb/demo/macros/macros

Breakpoint 1, do_sumfing () at macros.c:7
7               printf("here.\n");
(gdb) macro expand I_HAS_A_MACRO(variable)
expands to: printf("the value of x is %s\n", variable);
(gdb) info macro BLEH
The symbol 'BLEH' has no definition as a C/C++ preprocessor macro
at /home/michael/NoName/gdb/demo/macros/macros.c:7
(gdb) info macro LOKL_MACRO
Defined at /home/michael/NoName/gdb/demo/macros/macros.c:6
#define LOKL_MACRO(y) printf("muh.\n");
```

EOF

Fragen?