Umeå University
Department of Applied Physics and Electronics

Linux as Development Environment 7.5 ECTS
5EL142 HT-16

Assignment 8

Submitted 2016-01-02
Author: Lorenz Gerber, 20161202-2033 (lorenzottogerber@gmail.com)
Instructor: Björne A. Lindberg

1 GDB

After download and untaring the archieve, I first compiled the program with *make* and run it. The screen output of the program showed two mathematical wrong terms. Second, I quickly inspected the source files, starting with *main.c*, then *gdblab.c*. Obviously, the output is generated by reading values from an array that is filled with values using a *for-loop*.

1.1 Changing Makefile

To debug, I added the *g* flag to the compile and link commands as shown below:

```
CC = gcc
LIBFLAG = -L. -lgdb -Wl,-rpath,.

all: program

program: main.c libgdb.so
$(CC) -g -o program main.c $(LIBFLAG)

libgdb.so: lib/gdblab.c lib/gdblab.h
$(CC) -g -c -fPIC lib/gdblab.c
$(CC) -g -shared -fPIC -o libgdb.so gdblab.o
```

1.2 Debugging

The debugger was started with *gdb ./program*. First the command *list* was used to get a quick view on the source code. Here it was seen that the code of interest is not within *main.c* but in the *test* function from the dynamically loaded library *gdblab.c*. Hence a break point for the *test* function was set by *break test*. Then the program was started with *run*. After hitting the breakpoint, *list* was used again to get an overview of the source code. Now *display test->buffert1* and *display test->buffert2* was used to get updated values for the respective variables in every step. Finally, using *next*, respectively the shortcut *n* was used to step through the rest of the code. After every step, the above mentioned variables/datastructures are presented on the screen.

1.3 Fixing the code

In the above described debugging session, it became obvious that the problem was a too small allocated array, *test->buffert1*. It had size 16. In each for loop from 0 to 16 the index value was assigned to the corresponding array index. However, 0 to 16 are 17 values. Hence the last value, which is needed in the following print statement, is written in the zero index position of *text->buffert2*. There it is overwritten later. Hence, the first print statement that accesses test->buffert1[16] actually accesses test->buffert2[0] with the value 45. This bug can be fixed by increasing the test->buffert1 array to size 17. The second print statement accesses the wrong array (*test->buffert1* instead of *test->buffert2*). The corrected source code is shown below:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
```

```
#include "gdblab.h"

int test()
{
int i;
typedef struct{
int buffert1[17];
int buffert2[16];
}theTest;

theTest *test = malloc(sizeof(theTest));

int j=2;

for (i=0; i<=16; i++)
{
test->buffert1[i] = i;
}

test->buffert2[0] = 45;
test->buffert2[1] = 5;

printf("16 + 14 = %i\n", test->buffert1[16] + test->buffert1[14]);
printf("45 + 5 = %i\n", test->buffert2[0] + test->buffert2[1]);
free(test);

//test->buffert2[0] = 5;
//printf("16 + 14 = %i\n", test->buffert1[16] + test->buffert1[14]);

//kill(getpid(),SIGSEGV);

return 0;
}
```

2 Trace

The program was downloaded and made executable by *chmod +x errorprog*. Then the program was run from the command line. On exit, *echo $?* returns 255. According to *www.tldp.org*, 'Advanced Bash Scripting', 'Appendix E. Exit Codes with Special Meanings', 255 is the 'out of range' exit code as exit takes only values between 0 to 255.

Running *strace -o output.txt ./errorprog* produces the following output:

```
execve("./errorprog", ["./errorprog"], [/* 74 vars */]) = 0
brk(NULL)                               = 0x1ce0000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f85f120a000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
```

```
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=130544, ...}) = 0
mmap(NULL, 130544, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f85f11ea000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1864888, ...}) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f85f0c1e00
mprotect(0x7f85f0ddd000, 2097152, PROT_NONE) = 0
mmap(0x7f85f0fdd000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
mmap(0x7f85f0fe3000, 14752, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f85f11e9000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f85f11e8000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f85f11e7000
arch_prctl(ARCH_SET_FS, 0x7f85f11e8700) = 0
mprotect(0x7f85f0fdd000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f85f120c000, 4096, PROT_READ) = 0
munmap(0x7f85f11ea000, 130544)          = 0
open("data.txt", O_RDONLY)              = -1 ENOENT (No such file or directory)
read(-1, 0x7ffeb604d8b0, 250)           = -1 EBADF (Bad file descriptor)
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 4), ...}) = 0
brk(NULL)                               = 0x1ce0000
brk(0x1d01000)                          = 0x1d01000
write(1, "Read \n", 6)                  = 6
close(-1)                               = -1 EBADF (Bad file descriptor)
exit_group(-1)                          = ?
+++ exited with 255 +++
```

It can be seen that *errorprog* tries to open a file name *data.txt*. Which initially results in a 'ENOENT', 'No such file or directory' error. The error value returned to bash however comes from the last command, *close* which tries to close the file again. The program has obviously no proper error handling in place as the first error on *open* does not lead to disregarding the *close* command which results in an 'EABADF', 'Bad file descriptor' error. By standard definition Unix C programs/functions return -1 on error. As -1 is not in the range 0 to 255, an exit code of 255 results in bash.

Now a file name *data.txt* with some text in it was created and the program was invoked again with *strace -o output.txt ./errorprog* producing the following *output.txt*:

```
execve("./errorprog", ["./errorprog"], [/* 74 vars */]) = 0
brk(NULL)                               = 0x23a8000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f92487f3000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=130544, ...}) = 0
mmap(NULL, 130544, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f92487d3000
```

```
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1864888, ...}) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f924820700
mprotect(0x7f92483c6000, 2097152, PROT_NONE) = 0
mmap(0x7f92485c6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
mmap(0x7f92485cc000, 14752, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f92487d2000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f92487d1000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f92487d0000
arch_prctl(ARCH_SET_FS, 0x7f92487d1700) = 0
mprotect(0x7f92485c6000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f92487f5000, 4096, PROT_READ) = 0
munmap(0x7f92487d3000, 130544)          = 0
open("data.txt", O_RDONLY)              = 3
read(3, "123\n", 250)                   = 4
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 4), ...}) = 0
brk(NULL)                               = 0x23a8000
brk(0x23c9000)                          = 0x23c9000
write(1, "Read 123\n", 9)               = 9
write(1, "\n", 1)                       = 1
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Now in the trace it can be seen that *open("data.txt", O_RDONLY)* returns 3. Looking up the man pages for the system call *open* shows that this function returns the assigned file descriptor on success. Hence the assigned file descriptor in this case is '3'. In the end, the program exits with exit code 0, the standard Unix C exit code for success.