Hunting segfaults

for beginners

Uwe Völker

XING AG

August 2012

What is a segfault?

- segfault = segmentation fault

What is a segfault?

- segfault = segmentation fault
- every process has memory pages
- these pages are mapped to physical memory

What is a segfault?

- segfault = segmentation fault
- every process has memory pages
- these pages are mapped to physical memory
- if you try to access an invalid address
- (or write to a protected address)

What is a segfault?

- segfault = segmentation fault
- every process has memory pages
- these pages are mapped to physical memory
- if you try to access an invalid address
- (or write to a protected address)
- BOOOM!

Examples - C

- using uninitialized pointers
- dereferencing NULL pointers
- using "freed" pointers

Examples - Perl

- bug in a XS extension
- bug in Perl itself (rare)

Examples - Perl

- bug in a XS extension
- bug in Perl itself (rare)
- Perl 5.6.1:
- perl -e 'undef a'
- perl -e '*::=%::=0'

Examples - Perl

- bug in a XS extension
- bug in Perl itself (rare)
- Perl 5.6.1:
- perl -e 'undef a'
- perl -e '*::=%::=0'
- Perlmonks thread: (Golf) Segfault Perl
- `http://perlmonks.org/?node_id=156461`

On the shell

```
perl segfault.pl
Segmentation fault (core dumped)
```

On the shell

```
perl segfault.pl
Segmentation fault (core dumped)
```

```
#!/usr/bin/perl
use Debug::DumpCore;
Debug::DumpCore::segv;
```

Core dump file

```
$ ulimit -c unlimited
$ perl segfault.pl
Segmentation fault (core dumped)
$ ll core
-rw-r----- 1 uwe uwe 1695744 Jul 26 14:08 core
```

CGI script

- personal story: CGI script in Apache
- no output, no entry in logfiles (access.log and error.log)

CGI script

- personal story: CGI script in Apache
- no output, no entry in logfiles (access.log and error.log)
- but when I wrote to some file, the content was there
- so the script was getting executed...

Usage

- "Print out each line before it is executed (like `sh -x`)"

Usage

- "Print out each line before it is executed (like `sh -x`)"
- `perl -d:Trace program`
- for CGI: put it in your shebang line

Usage

- "Print out each line before it is executed (like `sh -x`)"
- `perl -d:Trace program`
- for CGI: put it in your shebang line

```
>> ./test:4:  print "Statement 1 at line 4\n";
>> ./test:5:  print "Statement 2 at line 5\n";
>> ./test:6:  print "Call to sub x returns ", &x(),
>> ./test:12:     print "In sub x at line 12.\n";
>> ./test:13:     return 13;
>> ./test:8:  exit 0;
```

How do I spot a segfault?

- look at the last few lines
- if it stops immediately, it might be a segfault

How do I spot a segfault?

- look at the last few lines
- if it stops immediately, it might be a segfault
- grep for your script name
- output can be very large, with long lines
- `grep -v site_perl`

How do I spot a segfault?

- look at the last few lines
- if it stops immediately, it might be a segfault
- grep for your script name
- output can be very large, with long lines
- `grep -v site_perl`
- in my case: buggy MSSQL driver (easysoft)

Other uses for Devel::Trace

- your program is behaving strange and you have no debugger at hand
- (use `grep` and `grep -v` to filter the output)

Other uses for Devel::Trace

- your program is behaving strange and you have no debugger at hand
- (use `grep` and `grep -v` to filter the output)
- Does this code get executed?
- Which part of the conditional was taken?

Introduction

- GNU debugger
- command line debugger
- we use it to extract the stacktrace from the core dump file

Usage

```
$ gdb perl core
Core was generated by `perl perl/segfault.pl'.
Program terminated with signal 11, Segmentation fau
#0  0x00007f2f5d086754 in crash_now_for_real (suicid
10      printf("%d", *p); /* cause a segfault */
(gdb)
```

Usage

```
$ gdb perl core
Core was generated by `perl perl/segfault.pl'.
Program terminated with signal 11, Segmentation fau
#0  0x00007f2f5d086754 in crash_now_for_real (suicid
10      printf("%d", *p); /* cause a segfault */
(gdb) where
#0  0x00007f2f5d086754 in crash_now_for_real (suicid
#1  0x00007f2f5d086789 in crash_now (suicide_messag
#2  0x00007f2f5d086820 in XS_Debug__DumpCore_segv (
#3  0x0000000000488db3 in Perl_pp_entersub ()
#4  0x000000000480a7d in Perl_runops_standard ()
#5  0x00000000004336b4 in perl_run ()
#6  0x000000000041bddc in main ()
(gdb)
```

Core dump file - reloaded

- `ulimit -c unlimited`

Core dump file - reloaded

- `ulimit -c unlimited`
- current directory has to be writable
- (can be tricky with Apache)

Core dump file - reloaded

- `ulimit -c unlimited`
- current directory has to be writable
- (can be tricky with Apache)
- `ps auxww|grep apache`
- `ls -l /proc/1234/cwd`

Usage

- "Automatic gdb backtraces on errors"

Usage

- "Automatic gdb backtraces on errors"
- just use the module
- it registers signal handlers for SIGSEGV (and a few more)

How does it work?

- the signal handler forks off a process which runs gdb
- gdb attaches to the parent and outputs the stacktrace

Sources

- http://en.wikipedia.org/wiki/Segmentation_fault
- http://modperlbook.org/html/21-6-Analyzing-Dumped-core-Files.html
- http://www.linux-magazin.de/Heft-Abo/Ausgaben/2007/01/Getriebeschaden

Questions?