Advance Techniques and Tools

Raphael Ahrens

12. Oct. 2012

Contents

What are Bugs?

9/9

0800 Arctan started
1000 " stopped - arctan ✓
{ 1.2700 9.037 847 025
9.037 846 ?95 correct
13"UC (032) MP - MC ~~1.9821447000~~ ~~2.13047764915 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.13047 6415
correct 2.13067 6415
Relays 6-2 in 033 failed special speed test
In relay " 10.00 test.
Relays changed
Relay 2145
Relay 3370
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 
Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Why do we have bugs?

Donald Knuth (1977)

Beware of bugs in the above code;
I have only proved it correct, not tried it.

Types of Bugs

Common Bugs

Common bugs are typical caused by ambiguous designs or holes in the test cases. They are easily reproducible if you find the right test case.

Sporadic Bugs

These bugs have the tendency to strike in unpredictable ways and you have to wait for it to happen. If you are confronted with this kind of bug you need to use loggers to figure out what is going on. Sometimes it takes a lot of time to gather the necessary data to figure out what is going on.

Example

```
key=0x42420000 + 0x100* uid + (timeInSecs) % 3600*24) #bug

key=0x42420000 + 0x100* pid + (timeInSecs) % (3600*24))#fix?
```

Heisenbugs

These bugs can really drive you mad. Under normal conditions they can easily be observed. But as soon as you start with a debugger or simply change the source code they vanish.

Types of Heisenbugs

- Race conditions
- Memory miss use
- False performance tunning
- ...

Bugs behind bugs

Sometimes two or more bugs only produce errors when they were both triggered. This is a special case in debugging, because we start to search for one bug. But when we find and fix this one bug it still does not work and we start searching again.

What is debugging?

Wikipedia

*Debugging is a **methodical** process of **finding and reducing** the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it **behave as expected**.[?]*

13 "Golden" Rules

1. Understand the requirements
2. Make if fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate facts from interpretation
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didn't fix it, it ain't fixed
13. Cover your bugfix with a regression test

Understand the requirements

Make sure that you have a reasonable understanding of what the problem is supposed to do and read the available documentation.

Make if fail

Create a test case where the bug is reproducible or at least find out what is necessary to reproduce the bug. Also only with a test case it is possible to ensure that the bug was fixed.

Simplify the test case

You need to reducing the number of influences nearly to the real cause of the problem.

Read the right error message

This is trivial but sometimes bug reports only mentions the result of a bug that is only the result of a chain reaction.

Check the plug

Check for basic error sources, like a full hard drive, missing network connection, wrong libraries and so on.

Separate facts from interpretation

Make clear what you really know and what is only your expectation. Try to find prove for your facts.

Divide and conquer

1. List all the parts that are involved in producing that bug.
2. Find tests to separate the working parts from the buggy ones.
3. If you think you know were the problem is try to conquer it.
4. If you didn't fixed it repeat at step 1.

Example

- divide environmental influences from source code changes

Match the tool to the bug

Use the right tool for the right job. You should familiarize yourself with the different tools, so you are prepared for the worst case.

One change at a time

Change only one thing and than test if it solves the problem and revert the changes if it didn't. There is one exceptions, if you have a bug which is hiding behind another bug.

Keep an audit trail

Keep track of your changes over the whole debugging session. Sometimes you need to test out different combinations of input oder code changes and it is a nightmare if you get lost.

If you didn't fix it it ain't fixed

If the bug doesn't show up after some changes and you can't say why, then you have to make sure that it is really fixed. If it is fixed and in most cases it isn't then at least find out why it was fixed.

Cover your bugfix with a regression test

The bug needs to be documented and the best way to do this is by creating a regression test.

Source Code Debugger

What is a Source Code Debugger?

With a source code debugger it is possible to step through a program step by step and visualize its state.

Some examples

- gdb (GNU)
- Visual Studio (Microsoft)
- LLDB (LLVM)
- ...

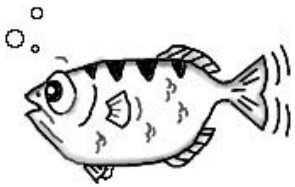
What can these tools do?

- display the data of the program
- display code that was executed
- stop the program at an interesting point
- step through the code

gdb

History

gdb was written by Richard Stallman in the year 1986 and since then it is under constant development.



Debug flag -g

gcc -g -o example example.c

Starting gdb

gdb <program>
gdb –args <program> <arg1> <arg2>

	command	description
running		
	run <args>	run the program
	start <args>	start the programm and break at start
data		
	print <expr>	print the expression once
	display <expr>	print the expression with every step
code		
	backtrace	show the backtrace
	list	show the source listing

	command	description
stopping		
	break <file>:<line>	set a breakpoint
	watch <file>:<line>	show the
stepping		
	next	go to next line in the source file
	step	step into the next line
	finish	return from the current function
	cont	continue running

Memory Debugger

What can a memory debugger find?

- Memory leaks
- Buffer overflows
- Uninitialized memory
- Incorrect use of memory

valgrind

What is valgrind?

Valgrind is a GPL system for debugging and profiling Linux programs.

The toolbox

- **Memcheck**
- Callgrind
- Massif
- Cachegrind
- Helgrind
- ...

Usage

valgrind - -tool <name>

What does Memcheck do?

It interprets the compiled machine code and keeps track of all the memory operations, like allocating memory, freeing memory and access to memory .

Things it can detect

- Double frees
- Illegal read/writes
- Use of uninitialised memory
- Memory leaks
- ...

Static Code Checker

What is a static code analyzer?

Static code analyzers check the source code for common errors without executing the code. Instead they only look at the source code.

How they should be used

A static checker does not understand the program logic, so it is possible that it detects errors that are not errors. The goal is to reduce the checker output to a minimum.

Tools

- gcc -Wall -Werror
- lint <source file>
- splint <source file>
- ...

[] Debugging @ONLINE, 2012. URL http://en.wikipedia.org/w/index.php?title=Debugging&oldid=510908542.