Debugging and Profiling

Software Engineering 2008
ETH Zurich

(many examples taken from “Andreas Zeller: Why programs fail.”)
Overview

• Context
• Debugging: Activity, Strategy, Tools
• Profiling: Activity, Strategy, Tools
• Summary
Part I:
Debugging
Your program crashes

Short description
The application KPackage (kpackage) crashed and caused the signal 11 (SIGSEGV).

What is this?
An application mostly receives the SIGSEGV signal due to a bug in the application. The application was asked to save its documents.

What can I do?
You might want to send a bug report for this application. Check if it is listed on http://bugs.kde.org, otherwise mail the author. Please include as much information as possible, maybe the original documents. If you have a way to reproduce the error, include this also.
0800  Andam started  \{ 1.2700  9.037 872 025
1000  stopped - andam \checkmark
13 Vc (032) MP - MC 2.150776495(2) 4.615925059(2)
(033) PRO 2 2.130476415
cond 2.130676415
Relays 6-2 in 033 failed special speed test
In relay
Relays changed
1100  Started Cosine Tape (Sine check)
1525  Started Multi-Adder Test
1545  Relay #70 Panel F
      (moth) in relay.
First actual case of bug being found.
1600  Andangt started.
1700  closed down.
Debugging

• Structured activity to find the cause for abnormal program behavior (a.k.a. “bugs”)

• Structure:

  A) **Locate** the faulty source code
  B) **Fix** the source code (without creating new bugs)
  C) **Learn** from the bug and **apply** what you have learned
Traffic Schema

• **T**rack the problem
• **R**eproduce
• **A**utomate
• **F**ind Origins
• **F**ocus
• **I**solate
• **C**orrect
Locate the faulty source code
Location strategies

• Narrow down the input
• Observe the dynamic state
• Simplify the environment
Infected States

Error code is executed

Inconsistent state

Error becomes visible

$t$
Scientific Method

- Observe (or have someone else observe) some aspect of the universe.
- Invent a tentative description, called the hypothesis, that is consistent with the observation.
- Use the hypothesis to make predictions.
- Test those predictions by experiments or further observations and modify the hypothesis in the light of your results.
- Repeat steps 3 and 4 until there are no discrepancies between hypothesis and experiment and/or observation.

From: A. Zeller: Why programs fail
Scientific Method for Debugging

- Observe a failure.
- Invent a hypothesis as to the failure cause that is consistent with the observation.
- Use the hypothesis to make predictions.
- Test the hypothesis by experiments and further observations:
  - If the experiment satisfies the prediction, refine the process.
  - If the experiment does not satisfy, create an alternate hypothesis.
- Repeat steps 3 and 4 until the hypothesis can no longer be refined.
Divide et impera

- Basis for tracking the problem
- Isolate causes
Narrow down the input

double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Narrow down the input

double bug(double z[], int n) {
    int i,j;
    
    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Narrow down the input

double bug(double z[], int n) {
    int i,j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Narrow down the input

double bug(double z[], int n) {
    int i,j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }

    return z[n];
}
double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Narrow down the input

double bug(double z[], int n) {
    int i, j;

    i = 0;
    for (j = 0; j < n; j++) {
        i = i + j + 1;
        z[i] = z[i] * (z[0] + 1.0);
    }
    return z[n];
}
Delta Debugging

• Automate the process
  – Cut the input set into parts
  – Tests for robustness: only few inputs are rejected, many do not create problems
Gecko Bug Report

Mozilla bug report #24735:

Ok the following operations cause mozilla to crash consistently on my machine

→ Start mozilla
→ Go to bugzilla.mozilla.org
→ Select search for bug
→ Print to file setting the bottom and right margins to .50 (I use the file /var/tmp/netscape.ps)
→ Once it’s done printing do the exact same thing again on the same file (/var/tmp/netscape.ps)
→ This causes the browser to crash with a segfault
BugAThon Instructions

Start removing HTML markup, CSS rules, and lines of JavaScript from the page. Start by removing the parts of the page that seem unrelated to the bug. Every few minutes, check the page to make sure it still reproduces the bug. [. . . ] When you’ve cut away as much HTML, CSS, and JavaScript as you can, and cutting away any more causes the bug to disappear, you’re done.
<table>
<thead>
<tr>
<th>op_sys</th>
<th>priority</th>
<th>bug_severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 3.1</td>
<td>P1</td>
<td>blocker</td>
</tr>
<tr>
<td>Windows 95</td>
<td>P2</td>
<td>critical</td>
</tr>
<tr>
<td>Windows ME</td>
<td>P3</td>
<td>major</td>
</tr>
<tr>
<td>Windows 2000</td>
<td>P4</td>
<td>normal</td>
</tr>
<tr>
<td>Windows NT</td>
<td></td>
<td>trivial</td>
</tr>
<tr>
<td>Mac System 7</td>
<td></td>
<td>enhancement</td>
</tr>
<tr>
<td>Mac System 8.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mac System 8.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mac System 9.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MacOS X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BeOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-UX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutrino</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenVMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SunOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BeOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-UX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutrino</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenVMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SunOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenBSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BeOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-UX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutrino</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenVMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SunOS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Delta Debugging HTML

```html
<SELECT NAME="priority" MULTIPLE SIZE=7> ✘
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✘
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
```
Delta Debugging HTML (cont.)

```html
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✘
<SELECT NAME="priority" MULTIPLE SIZE=7> ✘
<SELECT NAME="priority" MULTIPLE SIZE=7> ✘
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✔
<SELECT NAME="priority" MULTIPLE SIZE=7> ✘
```
Gecko Bug Report Improved

Printing a page containing <SELCT> makes Mozilla crash.
Observing the dynamic state

- “printf” + log files
- Debuggers
  - Slow down and/or stop the execution of the program
  - Display content of variables
  - Analyze expressions and graph structures
- Observe the state at the right level of abstraction!
Using printf

- printf (or print, system.out.println, WriteLn, etc.) prints a message to the standard out.
- Make code execution visible.
- Use 'stderr' variants to prevent mixing regular and debug output.
- Print to a file for non-console applications (log files).
- Implement a 'verbose' infrastructure.
Using a debugger

```c
struct x {
    int value;
    struct x *next;
};

int main (int argc, char** argv)
{
    struct x *p1;
    struct x *p2;
    struct x *p3;
    struct x *p4;

    p1 = malloc(sizeof(struct x));
    p2 = malloc(sizeof(struct x));
    p3 = malloc(sizeof(struct x));
    p4 = malloc(sizeof(struct x));

    p1->value = 10;

    (gdb) graph display *(p1->next->next) dependent on 3
    (gdb) graph display *(p1->next->next->next) dependent on 4
    (gdb) graph display *(p1->next->next->next->next) dependent on 5
    (gdb) :Disable display 6 to avoid infinite recursion.
    (gdb)

    In display 2: p1->next (double-click to dereference)
```
EiffelStudio debugger

DEMO
Some debugging terminology

- Debug symbols
- Breakpoint
- Step
- Stack trace
- Reverse execution
- Watch expression
Why things get more complicated with object-orientation

SET [X]

SET_A [X]

SET_B [X]
deferred class SET[X]

[...]

feature – Comparison
    is_subset_of (other: SET[X]): BOOLEAN is
        -- Is a subset of `other'? deferred
    end

    is_superset_of (other: SET[X]): BOOLEAN is
        -- Is a superset of `other'? deferred
    end

[...] end
class SET_A[X] 
inherit SET[X] 
[...]

feature – Comparison
  is_subset_of (other: SET[X]): BOOLEAN is
    -- Is a subset of `other'?
    do
      Result := other.is_superset_of (Current)
    end
  is_superset_of (other: SET[X]): BOOLEAN is
    -- Is a superset of `other'?
    deferred
      [some implementation of superset]
    end
[...] 
end
Code of SET_B[X]

class SET_B[X]
inherit SET[X]

[...]

feature – Comparison
  is_subset_of (other: SET[X]): BOOLEAN is
    -- Is a subset of `other'?
    do
      [some implementation of subset]
    end
  is_superset_of (other: SET[X]): BOOLEAN is
    -- Is a superset of `other'?
    deferred
      Result := other.is_subset_of (Current)
    end

[...]
end
Does it work?

local
  a: SET_A[INTEGER]
  b: SET_B[INTEGER]
do
  create a
  create b
  print (a.is_subset_of (b))
end
Fix the source code
Fixing the code

- **Patch / Hot Fix**
  - Changes the code to at the location where the error manifests.
  - Small changes.

- **Fix / Correction**
  - Remove the error from the code

- **Re-design / Re-implemetation**
  - Change large portions of the code to remove the error.
When you fix code

• **Really fix it!**

• Remember:
  – each fix is a change
  – *every change may create bugs*

• Sometimes: a fix creates a bug creates a fix creates a bug creates a fix
  – Probably a Design error!

• If it is not your code: notify “upstream”
OpenSSH 5.0 has just been released. [...] We apologise for any inconvenience resulting from this release being made so shortly after 4.9. Unfortunately we only learned of the below security issue from the public CVE report. The Debian OpenSSH maintainers responsible for handling the initial report of this bug failed to report it via either the private OpenSSH security contact list (openssh@openssh.com) or the portable OpenSSH Bugzilla (http://bugzilla.mindrot.org/).
[..] looking into the patch and the errata release is the exact same as RedHat had done 2 years and 8 months ago here:

Learn from the bug and apply what you have learned
 SOFTWARE ENGINEERING COMIX #1

Gosh darn, another BUG! What's wrong with you programmers, anyway?

AW, BOSS, THERE HAVE ALWAYS BEEN BUGS, AND THERE ALWAYS WILL BE. NOTHING WE CAN DO ABOUT IT. NOW, I'LL FIX THIS ONE IN A JIFFY.

Not so fast, young programmer. Have you ever heard of The Three Questions?

OLD GEEZER. PROBABLY STILL WRITES IN FORTRAN.

NO, IS IT A ROCK GROUP?

YOU CAN WRITE BETTER PROGRAMS WITH LESS EFFORT IF YOU ASK YOURSELF THREE QUESTIONS EVERY TIME YOU FIND A BUG.

The first question is: Have I made this error anywhere else?

I GUESS I SHOULD CHECK THE OTHER INDEX CALCULATIONS.

The second question is: What happens when I fix the bug?

WILL MY FIX CAUSE A BUG LATER?

The third question is: How can I change my ways to make this kind of bug impossible?

THIS STUFF SHOULD REALLY BE DONE IN A DEFINE.

Hi-yo Pascal, awaaaaah!

WHO WAS THAT OLD HACKER ANYWAY?
How to apply your findings

• Do extensive checking in libraries
  – Add assertions/contracts
  – Improve typing

• Define coding standards

• Improve coding standards and libraries when you find bugs

• Enforce assertions/typing/coding standards by tools
#include <stdio.h>

int main (int argc, char **argv)
{
    char a[25];

    strcpy (a, "Hello world");
    printf ("%s\n", a);
    return 0;
}
/tmp/ccv14745.o(.text+0x25): In function `main':
  warning: strcpy() is almost always misused, please use strlcpy()
Part II: Profiling
Profiling

- Analyze a program.
- Observe resources:
  - Call frequency
  - Overall time
  - Memory
  - CPU time
  - Disk-space
  - Energy consumption
- Most often: minimize resource usage
Heisenberg

Profiling often suffers from a “Heisenberg”-like effect:

1. Profiling means executing extra code
2. Extra code needs time
3. Change in the results of measurement
Types of profilers

- Manual (think: printf)
- Compiler assisted: EiffelStudio
- Runtime: Valgrind
- Hypervisor (new)
EiffelStudio profiling

DEMO