Debugging, Linting, Grinding

SW Dev 2020
Lecture 5

92

9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
~~1.9621442000~~
13"uc (032) MP - MC ~~2.130476415 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.

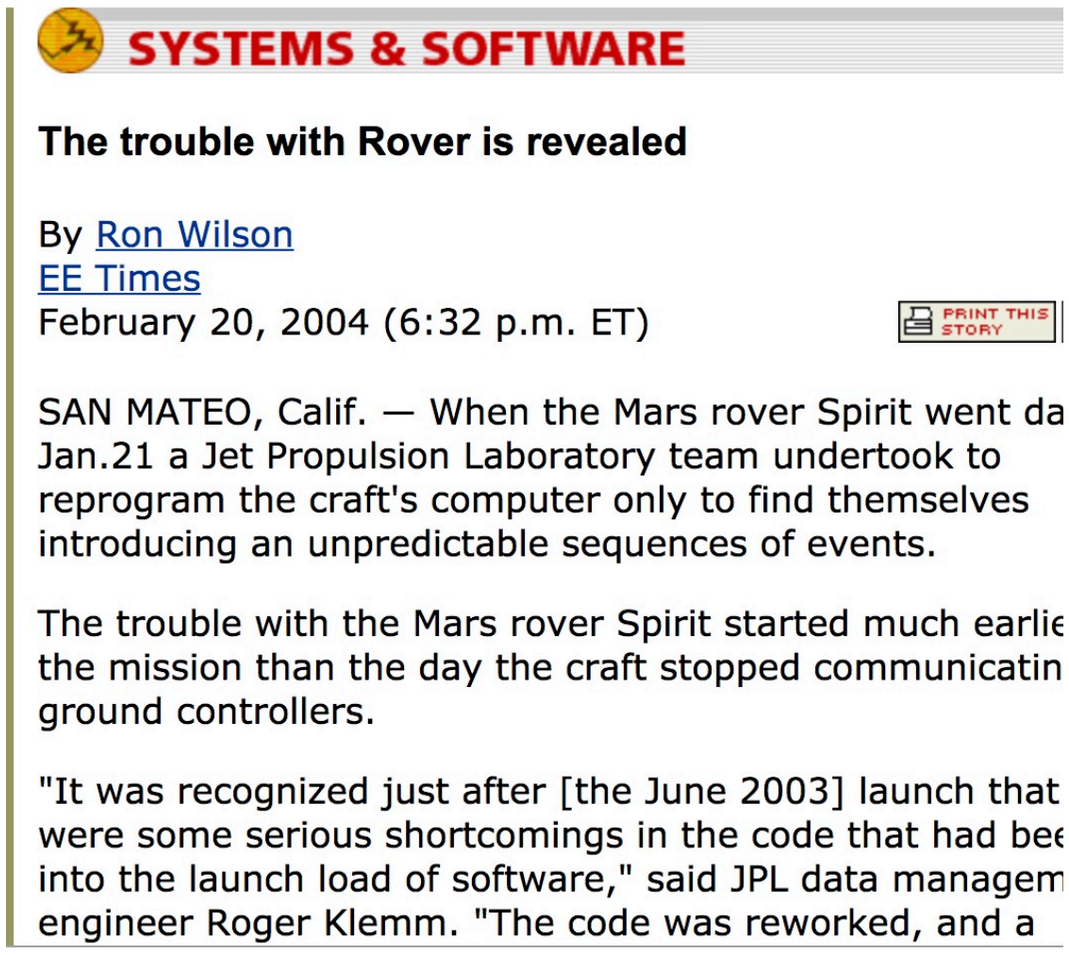
Relay 2145
Relay 3370

1545 
Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.

~~1700~~ 1630 Arctangent started.
1700 closed down.

How do we debug?

- What tools do we have?
- Depends on the situation!
  - BIOS — oscilloscopes
- We’re comparatively lucky



SYSTEMS & SOFTWARE

The trouble with Rover is revealed

By Ron Wilson
EE Times
February 20, 2004 (6:32 p.m. ET) PRINT THIS STORY

SAN MATEO, Calif. — When the Mars rover Spirit went da
Jan.21 a Jet Propulsion Laboratory team undertook to
reprogram the craft's computer only to find themselves
introducing an unpredictable sequences of events.

The trouble with the Mars rover Spirit started much earlie
the mission than the day the craft stopped communicatin
ground controllers.

"It was recognized just after [the June 2003] launch that
were some serious shortcomings in the code that had bee
into the launch load of software," said JPL data managem
engineer Roger Klemm. "The code was reworked, and a

Rebooting on Mars

Matthew Fordahl, The Associated Press 2004-02-15

It's a PC user's nightmare: You're almost done with a lengthy e-mail, or about to finish a report at the office, and the computer crashes for no apparent reason. It tries to restart but never quite finishes booting. Then it crashes again. And again.

Getting caught in such a loop is frustrating enough on Earth. But imagine what it's like when the computer is 200 million miles away on Mars. That's what mission controllers faced when the Mars rover Spirit stopped communicating last month.

Ultimately, the fix that saved Spirit wasn't that different from how a PC would be repaired on Earth. It's just that the folks who have their hardware on Mars -- and the eyes of the world on them -- are better prepared for disaster.

Tech support for an $820 million mission is a cautious affair. Tools to recover from and fix any problem must be built into the system before launch. The systems' behaviors need to be completely understood and predictable.

"Luckily, during the design period, we anticipated that we might get into a situation like this," said Glenn Reeves, who oversees the software aboard the Mars rovers Sprit and Opportunity at NASA's Jet Propulsion Laboratory.

For stability, reliability and predictability, mission designers did not bust the budget and design the hardware or software from scratch. Instead, they turned to hardware and software that's been used in space before and has a proven track record on Earth as well.

"The advantage of using commercial software is it's well-known, and it's well deployed," said Mike Deliman, an engineer at Alameda-based Wind River Systems Inc., which made the rovers' operating system. "It has been used throughout the world in hundreds of thousands of applications."

The operating system, VxWorks, has its roots in software developed to help Francis Ford Coppola gain more control over a film editing system. But the developers, David Wilner and Jerry Fiddler, saw a greater potential and eventually formed Wind River, named for the mountains in Wyoming. VxWorks became a formal product in 1987.

The operating system is embedded in systems that control jetliners and atomic colliders, anti-lock braking systems in cars and even heart pacemakers. It's also been used successfully in the Mars Pathfinder lander, Mars Odyssey orbiter and Stardust comet probe.

"These are all things that can't afford to fail," Deliman said.

We have help!

- Hardware support (code & data breakpoints, `trap`)
- Language support (Type systems
- IDE & Tooling (Syntax checking

Coding Style Guidelines

- Google has one
- CMU SEI has one
- Nasa has one
- We have one — (CwC!)

Print-line debugging

The "Wolf Fence" Algorithm for Debugging

Edward J. Gauss
University of Alaska

The "Wolf Fence" method of debugging time-sharing programs in higher languages evolved from the "Lions in South Africa" method that I have taught since the vacuum-tube machine language days. It is a quickly converging iteration that serves to catch run-time errors.

The same faulty thinking that produced the error in the first place may recur during the use of dumps. If one knows where the error is located, a dump can be quite useful. The "Wolf Fence" method compels attention to that portion of the program containing the error. It is described as follows: (1) Somewhere in Alaska there is a wolf. (2) You may build a wolf-proof fence partitioning Alaska as required. (3) The wolf howls loudly. (4) The wolf does not move.

The procedure is then:

1. Let A be the territory known to contain the wolf (initially all of Alaska).
2. Construct a fence across A, along any convenient natural line that divides A into B and C.
3. Listen for the howls; determine if the wolf is in B or C.
4. Go back to Step 1 until the wolf is contained in a tight little cage.

Any convenient PRINT instruction will serve as a "wolf fence." It must display its location in order to identify its output uniquely, e.g.,

```
PRINT, "Wolf fence at line 1234"
```

The program is run and the output examined. The "howls of the wolf," the error indication, will be found in either the territory before or after the fence. Additional fences are constructed until the programmer clearly sees the exact location of the error. Convergence can be accelerated by the addition of several fences per iteration.

The best location for fences is after the label of any program segment. Both Cobol and Pascal are written so that a fence can also be conveniently placed after the label but before the procedure. Fortran, BASIC, and APL can be written in this manner. In Fortran, a CONTINUE is the only command that may carry a statement number. In BASIC, a REM is used for an identified location, and in APL a lamp illuminates the entry to a procedure.

CR Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids*
General Terms: None
Additional Key Words and Phrases.
Author's present address: E.J. Gauss, American Bell, 307 Middleton-Lincroft Road, Lincroft, NJ 07738.

The "Wolf Fence" Algorithm for Debugging

Edward J. Gauss
University of Alaska

The "Wolf Fence" method of debugging time-sharing programs in higher languages evolved from the "Lions in South Africa" method that I have taught since the vacuum-tube machine language days. It is a quickly converging iteration that serves to catch run-time errors.

The same faulty thinking that produced the error in the first place may recur during the use of dumps. If one knows where the error is located, a dump can be quite useful. The "Wolf Fence" method compels attention to that portion of the program containing the error. It is described as follows: (1) Somewhere in Alaska there is a wolf. (2) You may build a wolf-proof fence partitioning Alaska as required. (3) The wolf howls loudly. (4) The wolf does not move.

The procedure is then:

1. Let A be the territory known to contain the wolf (initially all of Alaska).
2. Construct a fence across A, along any convenient natural line that divides A into B and C.
3. Listen for the howls; determine if the wolf is in B or C.
4. Go back to Step 1 until the wolf is contained in a tight little cage.

Any convenient PRINT instruction will serve as a "wolf fence." It must display its location in order to identify its output uniquely, e.g.,

PRINT, "Wolf fence at line 1234"

The program is run and the output examined. The "howls of the wolf," the error indication, will be found in either the territory before or after the fence. Additional fences are constructed until the programmer clearly sees the exact location of the error. Convergence can be accelerated by the addition of several fences per iteration.

The best location for fences is after the label of any program segment. Both Cobol and Pascal are written so that a fence can also be conveniently placed after the label but before the procedure. Fortran, BASIC, and APL can be written in this manner. In Fortran, a CONTINUE is the only command that may carry a statement number. In BASIC, a REM is used for an identified location, and in APL a lamp illuminates the entry to a procedure.

CR Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids*
General Terms: None
Additional Key Words and Phrases.
Author's present address: E.J. Gauss, American Bell, 307 Middleton-Lincroft Road, Lincroft, NJ 07738.

Interactive Debugging

- **GDB** is the venerated, venerable tool
- LLDB, etc are descendants
- Some w/GUI front-end
- Many bells, whistles: read the friendly manual

Automatic Tools

Static Analysis Tools

- Exercise all possible pathways through source
- Speedy
- Different analyses catch different classes of errors
- E.g. check conformance to style guidelines

Static Analysis, ctd.

- Conformity to coding standards, inconsistent formatting
- syntax errors
- undeclared variables
- Type mismatches

C++ Static Analysis Tools

- `flawfinder`
- `cppcheck`
- `clang-tidy`
- G++ & Clang++ address-sanitizer

Dynamic Analysis

Dynamic Analysis

- Exposes bugs too subtle for static analysis to find
- Execution of instrumented code
- Actual running of actual program

Valgrind

Current release: valgrind-3.10.1

Valgrind

- Memory analyzer
- Do not use with optimized builds (doesn’t work)
- Do not use with `-fsanitize=address`
- Slower execution, might notice on big, slow programs
- (Apparently) does not run under OS/X

DrMemory?

Heisenbugs

heisenbug: /hi:�zen�buhg/, n.

[from Heisenberg's Uncertainty Principle in quantum physics] A bug that disappears or alters its behavior when one attempts to probe or isolate it. (This usage is not even particularly fanciful; the use of a debugger sometimes alters a program's operating environment significantly enough that buggy code, such as that which relies on the values of uninitialized memory, behaves quite differently.) Antonym of *Bohr bug*; see also *mandelbug*, *schroedinbug*. In C, nine out of ten heisenbugs result from uninitialized auto variables, *fandango on core* phenomena (esp. lossage related to corruption of the malloc *arena*) or errors that *smash the stack*.

Heisenbugs - entry “Jargon File”

Heisenbugs, ctd.

- `for i in {1..1000}; do ./a.out; done;`

- “Chaos mode” in `rr`

Time traveling debugger

- Elm (Click)
- `rr` for C++
- IDE support:
  - VS Code
  - CLion
  - Emacs

Beyond …

- … Correctness — Profiling (time, space, hot-spot)
- … Interactivity — Automated bug-fixing!!! (Research)
- … Test suites —Fuzzing

And yet, there are still ***bugs***!

Formal Methods

- TLA+
- Liquid Haskell
- Verilog
- HOL (Light)
    - Pentium Bug, anyone?

Between Scylla and Charybdis

Demo GDB & Valgrind

Bonus: GDB Power Use Video