Debugging

Mathieu Lacage – DREAM
<lacage@sophia.inria.fr>

INRIA
SOPHIA ANTIPOLIS

Overview

- Debugging techniques.

- Demonstration of a debugging tools.

What is a bug ?

- The “unexpected” behavior of a system.
- There is a difference between:
  - how you expect the system to work
  - how the system works for real

- If these definitions do not match, you have a bug

How do you find its source ?

You need to understand how the system works for real.

A daunting task

- It is hard to know where to start: it depends on the system, on the tools you have, etc.
- The debugging methodology is always the same.

The rules

- The rules are easy to remember.
- Using the rules is hard but it is easy to try to use them:
  - reproduce the bug: understand what the user expects
  - understand the system
  - narrow the search
  - get a fresh view

Reproduce the bug

- Why ?
  - verify that the bug exists before
  - verify that the bug does not exist anymore after
  - understand bug parameters
- Write down:
  - the steps to reproduce the bug
  - what happens ?
  - what should happen ?
  - how often does it happen ?

Understand the system

- What is the system ?
  - the source code
  - the build tools
  - the run environment
- Read the manual!
  - build tools have great documentation
  - standard libraries have great documentation
  - developer documentation

Understand the system -- example

```
{
 char *str = g_strconcat (“one”,
   “two”, “three”);
}
```

The manual:

The variable argument list must end with NULL. If you forget the NULL, g_strconcat() will start appending random memory junk to your string.

Narrow the search

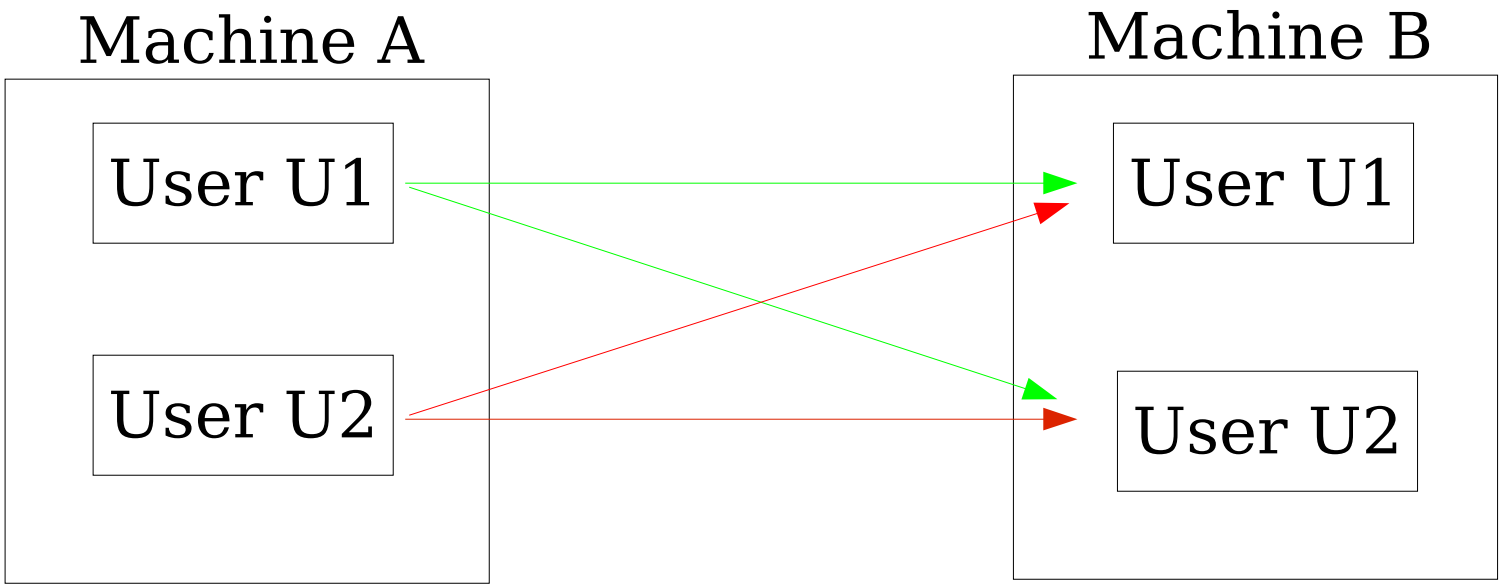
- Apply dichotomy: “Divide and Conquer”
  guess a number between 1 and 100
- Start from the failure
  ask yourself: “what fails” and not “what works”
- Change one thing at a time

Narrow the search – example 1

- Light bulb does not work:
  - ✗ replace it with a bulb which you know works
  - ✗ verify the wiring to the bulb
  - ✓ verify the wiring to the light switch
  - verify the wiring between the light switch and the light bulb

Narrow the search – example 2

- ProActive: ssh authentication fails sometimes.



The Client UC has a problem

Get a fresh view

- Show the bug to someone else.

- Do not explain where you think the bug comes from: do not “pollute” the newcomer.

Software tools

Make sure you understand their limitations

- Print statements
- Log files
- Debuggers
- Memory checkers

Print statements

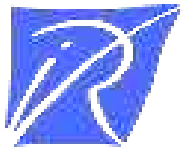
- Always include:
  - module name (filename, source line)
  - timestamp (date/time)
  - my_debug (“debug_string”);

- Formatting character strings can be CPU-costly (threads!).
- Sending the strings for on-screen display can VERY CPU-costly (threads!).

Log Files

- Follow the printf rules.
- Archive log files.
- Leave log file capability in production systems and disable it by default.

- Formatting the strings can be CPU-costly (threads!).
- Writing the files generates disk IO (threads!).



INRIA
SOPHIA ANTIPOLIS

Debuggers

- They work by placing breakpoints in the running assembly code.
- They have a deep influence on the system timing characteristics.
- They need “debugging information”: use the “-g” (g for debug, of course) compiler flag.
- They can:
  - step over code, functions
  - print backtraces
  - print variable content

INRIA
SOPHIA ANTIPOLIS

Memory Checkers: valgrind

- It ensures that each bit accessed in memory is:
  - correctly allocated
  - correctly initialized
  - correctly freed
- Limitations:
  - works on x86 processors only
  - requires huge amounts of RAM
  - is very very slow

References

- “Debugging: the 9 indispensable rules for finding the most elusive software and hardware problems” by David J. Agans

- Tool manuals
- Library manuals
- Developer documentation