Debugging & Bug-finding

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

D3S

Pavel Parízek

parizek@d3s.mff.cuni.cz

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Motivation

- When some test fails
  - You know there is a bug in the program code
  - You **do not** know the root cause of the bug
- Testing detects presence of bugs in the code
  - But you still have to find them and eliminate properly
  - Writing tests for smaller units of code does not help
    - Too much work with a little benefit (bad “cost-effect” ratio)
- Solution: debugging, automated bug-finders

Department of Distributed and Dependable Systems

Debugging

- Manual process
  - Monitoring execution of a given program
  - Inspecting and updating the current state
- Tool support
  - Stop and restart program execution
  - Manage breakpoints (set, delete)
  - Inspect and update memory content
    - e.g., the current values of program variables
  - Attach debugger to a running program

Important concepts

- Breakpoint
  - Source code location where the program execution is stopped intentionally
  - Additional conditions may have to be also satisfied
    - total number of hits, the current value of a program variable
  - Types: HW (CPU, fast, limited), SW (interrupt, slow)

- Core dump
  - Full memory image of the crashed process
    - heap objects and fields, registers, stack trace of each thread
  - Records the full program state upon crash

Department of Distributed and Dependable Systems

Basic approaches

- Printing debug messages
  - Add many `print` statements into your code
    - `System.out.println("[DEBUG] MyObj.doSmth: arg1 = " + arg1 + ", v = " + v + ", data = " + this.data);`
  - Read huge log files (search for text patterns)
  - Useful when you need lot of data at the same time

- Interactive "online" debuggers
  - Control program execution and inspect current state
  - Basic tools: **GDB**, DDD, jdb, JPDA, WinDbg, KD, CDB
  - IDE support: Visual Studio, Eclipse, NetBeans, IDEA

- Thorough explanation of your code to friends/colleagues
  - Approach works surprisingly well in practice

Department of Distributed and Dependable Systems

The [complete] process of debugging

Debugging bothers more than coding...

Six Steps of Debugging

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did it ever work???

Motivation to use debuggers

- John Carmack: Best programming setup and IDE | Lex Fridman Podcast Clips
  - https://www.youtube.com/watch?v=tzr7hRXcwkw
  - Length around 15 minutes

GNU Debugger (GDB)

GNU Debugger (GDB)

- User interface: command-line
- Intended for Unix-like systems
  - Low-level system software written in C/C++
    - Examples: utilities, web server, operating system kernel

- Supports many languages
    - **C**, **C++**, Rust, Ada, Go, Objective-C, ...

- Web site
  - http://www.sourceware.org/gdb/

Department of Distributed and Dependable Systems

Running program with GDB

- Start GDB for a given program

  `gdb <program>`
- Start program with arguments

  `gdb --args <program> <arg1> ... <argN>`
- Run program again inside GDB

  `(gdb) run [<arg1> ... <argN>]`

- Exit the debugged program

  `Ctrl+d (EOF)`
- End the GDB session

  `(gdb) quit`

Breakpoints

- Define breakpoint

```
(gdb) break <function name>
(gdb) break <line number>
(gdb) break <filename>:<line>
```

- Continue execution

```
(gdb) continue
```

  - Shortcut: `(gdb) c`

Department of Distributed and Dependable Systems

Breakpoints

- List of breakpoints

```
(gdb) info breakpoints
```

- Disable breakpoint

```
(gdb) disable <num>
```

- Enable breakpoint

```
(gdb) enable <num>
```

- Delete breakpoint

```
(gdb) delete <num>
```

Single stepping

- Advance to the next source line

  `(gdb) step [count]`

  - Shortcut: `(gdb) s`

- Advance to the next line in the current scope

  `(gdb) next [count]`

  - Shortcut: `(gdb) n`

Information about the debugged program

- Source code lines

```
(gdb) list
(gdb) list <linenum>
```

- Symbol table

```
(gdb) info scope <function name>
(gdb) info source
(gdb) info functions
(gdb) info variables
(gdb) info locals
```

Information about program variables

- Values

  ```
  (gdb) print <expression>
  ```

  - Example: `(gdb) print argv[1]`
  - Shortcut: `(gdb) p`

- Types

  ```
  (gdb) whatis <variable name>
  (gdb) ptype <variable name>
  ```

Inspecting the call stack frames

- Print call stack

  `(gdb) backtrace`

  - Shortcut: `(gdb) bt`
  - Including local variables

    `(gdb) bt full`

- Selecting frames
  - Move frame up: `(gdb) up [n]`
  - Move down: `(gdb) down [n]`

Department of Distributed and Dependable Systems

Changing expression values

- Make changes

```
(gdb) set var <expr> = <new value>
(gdb) print <expr> = <new value>
```

- Watch for changes (data breakpoint)

```
(gdb) watch <expression>
```

- List all watchpoints

```
(gdb) info watchpoints
```

Department of Distributed and Dependable Systems

Core dumps

- Set maximum size of core files

```
ulimit -c unlimited
```

- Analyze the core dump file ("core")

```
gdb <program binary> <core dump>
```

- Attach to already running process

```
gdb <program binary> <process ID>
```

Advanced features of GDB

- Calling functions and jumps
- Breakpoint command list
- Support for multi-threading
- Reverse execution
- Record and replay
- Remote debugging

- GUI frontend: DDD
  - http://www.gnu.org/software/ddd

Concurrency

- Debuggers support multi-threaded programs
  - Including GDB
- Problems
  - Programs behave differently when running in the debugger than in normal execution
    - Different internal timing of concurrent events
  - It is hard to find concurrency bugs with debuggers

Department of Distributed and Dependable Systems

Debugging tools for Windows/.NET

- Visual Studio debugger
  - Supported languages: C#, Visual Basic, ASP .NET
  - Advanced features: edit & continue, attach to running process, scriptability (reproduction of errors)
  - No support for debugging kernel space code
- Other tools
  - Windows debuggers (Windows SDK, WDK)
    - https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/
    - https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-operation-win8
    - Tools: WinDbg, KD, CDB, Psscor4, various utilities
- GDB-based: Visual Studio GDB Debugger, Visual GDB

Exercise

- Example
  - http://d3s.mff.cuni.cz/files/teaching/nswi154/sudoku.tgz
  - Build with Make (sets flags “`-g -Wall -O0`”)
  - Run via the command `./sudoku vstup.txt`

- Try basic features
  - Running the program in debugger
  - Management of breakpoints
  - Single stepping commands
  - Printing information about the program and variables
  - Inspecting the call stack and switching frames
  - Changing values of selected program variables

Automated run-time checking

- Idea: search for bugs during program execution
- Main approaches
  - Replacing libraries with debugging versions
    - Program linked with special versions of some library functions
    - Library functions (`malloc`, `free`, ...) perform runtime checks
    - Force program to crash upon a detected memory access error
    - Supported errors: buffer overflows, leaks, using freed memory
    - Tools: Dmalloc, DUMA
  - Monitoring execution of an instrumented program and looking for specific errors
    - Tools: **Valgrind**

Department of Distributed and Dependable Systems

Valgrind

- Generic framework for creating runtime checkers (error detectors)
  - Supported platforms
    - Linux: x86, x86-64, PowerPC
    - Android (x86, ARM), OS X
  - Basic principle: dynamic binary instrumentation

- Includes several tools
  - MemCheck: detects memory management errors
  - Helgrind: detects errors in thread synchronization

Running

- Command line:

```
valgrind <program> <arguments>
```

- Recommended compiler flags to use

```
-g -O0 -Wall -fno-inline
```

  - Avoid optimizations (`-O1`,`-O2`) when using Valgrind to detect errors in your program

MemCheck

- Running

```
valgrind [--tool=memcheck] <program>
```

- Supported errors
  - Accessing freed memory blocks
  - Reading uninitialized variables
  - Double-freeing of heap blocks
  - Memory leaks (missing "free")

- How to enable leak detection

```
valgrind --leak-check=yes <program>
```

MemCheck: output

- Buffer overflow

kind of error

stacktrace identifies the point where the error occurred

PID

```
== 2456 == Invalid write of size 4
== 2456 ==    at 0x204A68D: myfunc (myprog.c:95)
== 2456 ==    at 0x204A120: main (myprog.c:14)
== 2456 == Address 0x2684FF0 is 8 bytes after a block of
            size 64 alloc'd
== 2456 ==    at 0x2684FA8: malloc (vg_replace_malloc.c:130)
== 2456 ==    by 0x204A0E8: main(myproc.c:10)
```

description of the memory address involved in the error

- Memory leak

```
== 1789 == 32 bytes in 1 blocks are definitely lost in loss
   record 1 of 1
== 1789 ==    at 0x2F4482D: malloc (vg_replace_malloc.c:130)
== 1789 ==    at 0x204A692: myfunc (myprog.c:112)
== 1789 ==    at 0x204A130: main (myprog.c:20)
```

Department of Distributed and Dependable Systems

Issues

- Performance
  - Instrumented program runs 5-30 times slower than normal and uses much more memory
- Missed errors
  - Cannot detect off-by-one errors in the use of data allocated statically or on the stack
- Optimizations
  - Does not work well with -O1 and -O2

Department of Distributed and Dependable Systems

Exercise

- Try using Valgrind (MemCheck) on programs in the Linux distribution (`ls`, `cat`, ...) and on your simple programs in C/C++
  - Inspect reported warnings

Advanced topics

- Suppressions
  - Ignoring reported false positives and errors found in system libraries
- Useful options

  `--read-var-info=yes`
    - Information about variables (name, type, location)

  `--track-origins=yes`
    - Shows where the uninitialized variables come from
- Connecting Valgrind with GDB

Links

- GDB
  - http://www.sourceware.org/gdb
- jdb: The Java Debugger
  - https://docs.oracle.com/en/java/javase/17/docs/specs/man/jdb.html
- Dmalloc
  - http://dmalloc.com
- DUMA
  - http://sourceforge.net/projects/duma
- Valgrind
  - http://valgrind.org/
- Sanitizers from Google (address, memory, leak, thread)
  - https://github.com/google/sanitizers

Static code analyzers

- Automated search for common problems in source code at compile-time
  - bug patterns, suspicious constructs, bad practice
- Focus on semantics (behavior)
  - Compiler has already checked the syntax
- Modular analysis (each procedure separately)
- Trade-off: precision versus performance
  - false alarms (positives), missed errors

- Detect only simple bugs in the source code
  - but still very useful (highly recommended to use)

Department of Distributed and Dependable Systems

What the analyzers detect

- Basic patterns
  - Possible null dereferences
  - Comparing strings with ==
  - Ignoring result of method call
    - Example: `InputStream.read()`
  - Array index out of bounds
- Wrong usage of API
  - Stream not closed when exception occurs
- Memory usage errors
  - double `free()`, possible leaks

Tools

- Java
  - **SpotBugs** (FindBugs), Jlint, PMD, Checkstyle, Error Prone, **Checker Framework**
- C/C++
  - **Clang**, PRE*fast*, Cppcheck
- C#/.NET
  - StyleCop, FxCop, ReSharper, **Roslynator**
  - Microsoft Application Inspector
- Other (including commercial products)
  - SonarQube

SpotBugs / FindBugs

- Bug patterns detector for Java
- Source code available (LGPL)
- Historical context
  - FindBugs: original tool (research project), now abandoned
  - SpotBugs: recent fork, actively maintained, development
- Usage: command line, GUI, Ant, Maven, Gradle
- Integration with Eclipse (plugin)
- https://spotbugs.github.io/
- http://findbugs.sourceforge.net/

Demo: SpotBugs (FindBugs)

SpotBugs: advanced features

- Filtering bugs
- Annotations
- Data mining

Clang static analyzer

- LLVM compiler infrastructure project
- Clang front-end (C, C++, Objective-C)
- Source code available (BSD-like license)
- User interface: command-line
- http://clang-analyzer.llvm.org/

Department of Distributed and Dependable Systems

Demo: Clang

- Command: `scan-build`
  - Intercepts standard build process (CC, CXX)
  - Runs compiler and then static code analyzer

- How to use it
  - `scan-build` <*your build command*>
  - Examples
    - `scan-build ./configure ; make`
    - `scan-build gcc test.c mylib.c`

- Output: HTML files (bug reports)

Department of Distributed and Dependable Systems

Clang: options

- List all available checkers
  - Command: `scan-build -h`

- Enabling some checker
  - `scan-build -enable-checker [`*`name`*`]`

Department of Distributed and Dependable Systems

Exercise

- SpotBugs
  - Download and unpack
    - https://spotbugs.readthedocs.io/en/stable/
  - How to run it
    - Linux/Windows: `bin/spotbugs`
    - Other options (e.g., heap size)
      - https://spotbugs.readthedocs.io/en/stable/running.html

- Clang static analyzer

- Target programs
  - Your own (e.g., individual software projects)
  - Widely known open source software packages

Department of Distributed and Dependable Systems

Literature & additional information

- **The Debugging Book**
  - https://www.debuggingbook.org/
  - Basic introduction (overview)
  - Few selected advanced topics
    - Locating root causes of errors
    - Automatic repair (bugfixes)

- Diomidis Spinellis. **Modern Debugging: The Art of Finding a Needle in a Haystack.** Communications of the ACM, November 2018
  - https://cacm.acm.org/magazines/2018/11/232215-modern-debugging/
  - Guides on how to debug programs effectively and efficiently
  - Key point: use systematic approach instead of guessing
  - Key point: use advanced features of debugger tools (IDE)

Related courses

- Tools for detecting complicated bugs
  - concurrency (deadlocks, data races), assertions
  - NSWI101: Modely a verifikace chování systémů
  - **NSWI132: Analýza programů a verifikace kódu**

Department of Distributed and Dependable Systems

Links (other tools)

- Cppcheck: http://cppcheck.sourceforge.net/
- PMD: http://pmd.github.io/
- Checkstyle: https://checkstyle.sourceforge.io/
- Error Prone: http://errorprone.info/
- FxCop: https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-3.0/bb429476(v=vs.80)
- ReSharper: https://www.jetbrains.com/resharper/
- SonarQube: https://www.sonarqube.org/
- Microsoft Application Inspector
  - https://www.microsoft.com/security/blog/2020/01/16/introducing-microsoft-application-inspector/

Department of Distributed and Dependable Systems

Roslynator

- Extensible static analysis tool for C#
- Additional information
  - https://www.infoq.com/news/2020/01/roslynator-analyzers-231/
  - https://github.com/JosefPihrt/Roslynator
  - https://devblogs.microsoft.com/dotnet/write-better-code-faster-with-roslyn-analyzers/
  - https://learn.microsoft.com/en-gb/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019

Checker Framework

- Extends type system of Java
- Source code annotations
- Compiler plugins (“checkers”)
  - Responsible for type checking and inference

- Detects many kinds of bugs
  - null pointer exceptions, array index out of bounds, ...

- Web: https://checkerframework.org/

Homework

- Assignment
  - ReCodEx: group associated with this course
  - Web: http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/
- Deadline
  - 16.4.2025