Announcements / Agenda

- Finish / Review Threads Discussion
- HW 1 Performance
- HW 2 Questions
- Debugging Patterns
- GDB exercise
- Team Meetings
We could, for instance, begin with cleaning up our language by no longer calling a bug a bug but by calling it an error. It is much more honest because it squarely puts the blame where it belongs, viz. with the programmer who made the error. The animistic metaphor of the bug that maliciously sneaked in while the programmer was not looking is intellectually dishonest as it disguises that the error is the programmer’s own creation. The nice thing of this simple change of vocabulary is that it has such a profound effect: while, before, a program with only one bug used to be “almost correct”, afterwards a program with an error is just “wrong” (because in error).

Debugging, Analysis and Reverse-Engineering Environments

- gdb, jdb
- ptrace mechanism
- IDAPro
- Valgrind, Pin, DynamoRIO, Purify
- Gprof
- …many others
Building vs. Analysis

- Abstraction is a key design principle

- Abstraction is poison to the analysis & reverse engineering process

- Although automated regression testing is in some sense a debugging tool, I don’t really consider it a “debugging” activity in the following slides.
What is Debugging?

- The process of continuously diagnosing and replicating a feature or error in design, implementation, or environment to further our understanding of the flaw or the system as a whole.

- Detective work
  - The scientific process: continuously formulating hypotheses, designing repeatable experiments, and gathering evidence to support or refute our thesis.
The Science of System Design

- **OO principles**
  - Data hiding/encapsulation
  - Inheritance & composition, code reuse
  - Polymorphism

- **Design Patterns**

- **IDEs** (syntax highlighting, auto-refactoring, auto-completion), code libraries/APIs

- **UML, dataflow diagrams**

- **Model checking** (2008 ACM Turing Award)

- **Validation and regression testing**
The Art of Debugging

- No auto-analysis of comment quality
- External factors complicate analysis
  - “hidden” side effects in low-level code (IA-32)
  - Hardware caching effects (multiple levels)
  - Non-deterministic thread and process ordering
  - An instance of the Uncertainty Principle
- Lack of hardware support
  - Debugging & watchpoint registers limited
- Taught personally, experts pass on collected folk wisdom
  - Rarely about organized “testing”, test cases, and code review
Debugging Patterns

- Extraction & Production
- Control-based & Data-based
- Symbolic Execution
- Data organization & use patterns
- Impact analysis: cascading effects on data
- API knowledge vs. “Understanding Underneath”
- Trust Centers
- Generalizing Knowledge
  - What does a specific bug instance manifestation teach you about the system as a whole?
Extraction vs. Production

- Run programs in a debugger or IDE, not from console or desktop

- Programs should not ‘emit’ debugging information
  - Debugging environments and the code analyzer (human) should forcefully extract this information from the program by prodding it
GDB / DDD

- GDB = the GNU Debugger
  - jdb is similar tool for Java VM
  - DDD is a GUI for gdb
- Two major uses:
  - How do you know WHERE in control flow your program is?
  - How do you know state of CPU and important global and function-local information?
- How can you look at machine-level code?
GDB Exercise

- Run stockmarket in gdb
- Insert breakpoints at some_job and main
  - Examine registers and stack
- Insert a watchpoint on value of delta
- Manually change value of delta
- Observe number of threads running
- Observe change in program behavior under gdb