Comparing the Robustness of POSIX Operating Systems

http://www.ices.cmu.edu/ballista

Philip Koopman  &  John DeVale
ECE Department
koopman@cmu.edu - (412) 268-5225 - http://www.ices.cmu.edu/koopman

Institute for Complex Engineered Systems
DARPA
Carnegie Mellon
Overview: Ballista Automated Robustness Testing

- **Generic robustness testing**
  - Based on data types

- **OS Testing results**
  - Raw results for 15 Operating Systems
  - System calls vs. C Library

- **Exception Handling Diversity**
  - Does everyone core dump on the same exceptions? (no)

- **Approximating “Silent” failure rates (missing error codes)**

- **Conclusions/Future work**

A Ballista is an ancient siege weapon for hurling objects at fortified defenses.
Ballista: Software Testing + Fault Injection Ideas

- **SW Testing requires:**
  - Test case
  - Module under test
  - *Oracle* (a “specification”)

- **Ballista uses:**
  - “Bad” value combinations
  - Module under Test
  - *Watchdog timer/core dumps*

**Diagram:***

- **SPECIFIED BEHAVIOR**
  - SHOULD WORK
- **INPUT SPACE**
  - VALID INPUTS
- **RESPONSE SPACE**
  - ROBUST OPERATION
  - REPRODUCIBLE FAILURE
  - UNREPRODUCIBLE FAILURE

- **MODULE UNDER TEST**
  - UNDEFINED
  - SHOULD RETURN ERROR
  - INVALID INPUTS

- **Ballista combines ideas from:**
  - Domain testing ideas / Syntax testing ideas
  - Fault injection at the API level
Scalable Test Generation

**API**
write(int filedes, const void *buffer, size_t nbytes)

**testing objects**
- FILE
- DESCRIPTOR
- TEST OBJECT
- MEMORY
- BUFFER
- TEST OBJECT
- SIZE
- TEST OBJECT

**testing values**
- FD_CLOSED
- FD_OPEN_READ
- FD_OPEN_WRITE
- FD_DELETED
- FD_NOEXIST
- FD_EMPTY_FILE
- FD_PAST_END
- FD_BEFORE_BEG
- FD_PIPE_IN
- FD_PIPE_OUT
- FD_PIPE_IN_BLOCK
- FD_PIPE_OUT_BLOCK
- FD_TERM
- FD_SHM_READ
- FD_SHM_RW
- FD_MAXINT
- FD_NEG_ONE
- BUF_SMALL_1
- BUF_MED_PAGESIZE
- BUF_LARGE_512MB
- BUF_XLARGE_1GB
- BUF_HUGE_2GB
- BUF_MAXULONG_SIZE
- BUF_64K
- BUF_END_MED
- BUF_FAR_PAST
- BUF_ODD_ADDR
- BUF_FREED
- BUF_CODE
- BUF_16
- BUF_NULL
- BUF_NEG_ONE
- SIZE_1
- SIZE_16
- SIZE_PAGE
- SIZE_PAGEx16
- SIZE_PAGEx16plus1
- SIZE_MAXINT
- SIZE_MININT
- SIZE_ZERO
- SIZE_NEG

**test case**
write(FD_OPEN_RD, BUFF_NULL, SIZE_16)
CRASH Severity Scale

◆ **Catastrophic**
  - Test computer crashes (both Benchmark and Starter abort or hang)
  - Irix 6.2: \texttt{munmap( malloc((1<<30)+1), ((1<<31)-1)) );}

◆ **Restart**
  - Benchmark process hangs, requiring restart

◆ **Abort**
  - Benchmark process aborts (\textit{e.g.}, “core dump”)

◆ **Silent**
  - No error code generated, when one should have been
    (\textit{e.g.}, de-referencing null pointer produces no error)

◆ **Hindering**
  - Incorrect error code generated
Comparing Fifteen Operating Systems

Ballista Robustness Tests for 233 Posix Function Calls

- AIX 4.1
- Free BSD 2.2.5
- HP-UX 9.05
- HP-UX 10.20
- Irix 5.3
- Irix 6.2
- Linux 2.0.18
- LynxOS 2.4.0
- NetBSD 1.3
- OSF 1 3.2
- OSF 1 4.0
- QNX 4.22
- QNX 4.24
- SunOS 4.1.3
- SunOS 5.5

- Abort Failures
- Restart Failure

- 1 Catastrophic
- 2 Catastrophics
Failure Rates By POSIX Fn/Call Category
C Library Is A Potential “Robustness Bottleneck”

Portions of Failure Rates Due To System/C-Library

- AIX 4.1
- Free BSD 2.2.5
- HP-UX 9.05
- HP-UX 10.20
- Irix 5.3
- Irix 6.2
- Linux 2.0.18
- LynxOS 2.4.0
- NetBSD 1.3
- OSF 1 3.2
- OSF 1 4.0
- QNX 4.22
- QNX 4.24
- SunOS 4.1.3
- SunOS 5.5

Normalized Failure Rate

- System Calls
- C Library

- 1 Catastrophic
- 2 Catastrophics

Normalized Failure Rate

0% 5% 10% 15% 20% 25%
Common Failure Sources

◆ Based on correlation of failures to data values, not traced to causality in code

◆ Associated with a robustness failure were:
  • 94.0% of invalid file pointers (excluding NULL)
  • 82.5% of NULL file pointers
  • 49.8% of invalid buffer pointers (excluding NULL)
  • 46.0% of NULL buffer pointers
  • 44.3% of MININT integer values
  • 36.3% of MAXINT integer values

◆ Operational profile results vary depending on workload
  • IBS benchmarks: 19% to 29% weighted average failure rate
  • SPEC floating point less than 1% weighted average failure rate
Does Everyone Abort on the Same Things?

Number of versions compared at a time

QNX
FreeBSD
AIX
AIX + QNX
AIX + FreeBSD

LESS DIVERSE
MORE DIVERSE

168 POSIX Calls
17% (Normalized) Common Mode Aborts

Number of versions compared at a time

EVERY OS ABORTED ON THESE TESTS

168 POSIX Calls
Most System Call Aborts Potentially Avoidable

Number of versions compared at a time

△ 168 POSIX Calls
○ Only 72 System Calls

168 POSIX Calls
Only 72 System Calls
Data Analysis Using N-Version Detection

- **Use N-version software voting to refine data**
  (and use manual sampling to check effectiveness)
  - Eliminate non-exceptional tests -- **12% of data**; method ~100% accurate
    - *e.g.*, reading from read-only file
  - Identify Silent failures

- **Silent failures -- 6% to 17% additional robustness failure rate**
  - 80% accurate when one OS reports “OK” while at least one other OS reports an error code
    - ~2% were bugs involving failure to write past end of file
    - 28% of remainder due when POSIX permits either case
    - 30% of remainder due to false alarm error codes (*many in QNX*)
    - ~40% of remainder just out of scope of POSIX standard
  - 50% accurate when one OS reports “OK” but another OS dumps core
    - Half of remainder due to order in which parameters are checked
    - Half of remainder due to FreeBSD floating point library
      Abort failures (*e.g.*, fabs(DBL_MAX) )
Estimated Failure Rates After Analysis

Normalized Failure Rate by Operating System

- AIX
- FreeBSD
- HPUX 9.05
- HPUX 10.20
- Irix 5.3
- Irix 6.2
- Linux
- Lynx
- NetBSD
- OSF-1 3.2
- OSF-1 4.0
- QNX 4.22
- QNX 4.24
- SunOS 4.13
- SunOS 5.5

Normalized Failure Rate (after analysis)

Abort %
Silent %
Restart %
Catastrophic

Operating System Tested
Is Dumping Core The “Right Thing?”

- **AIX has only 10% raw Abort failure rate -- on purpose**
  - Wish to avoid Abort failures in production code
  - Ignores some NULL pointer reads by setting page 0 to read permission
  - *BUT -- 21% adjusted Abort failure rate; 12% Silent failure rate*

- **FreeBSD has 20% raw Abort failure rate -- on purpose**
  - Intentionally aborts to flag bugs during development cycle
  - 31% adjusted Abort failure rate; *BUT -- 17% adjusted Silent failure rate*

- **Future challenges:**
  - Flag defects during development
    - Boundschecker-like systems need a workload to find problems
  - And still tolerate robustness problems once system is fielded
    - Truly Portable exception handling for POSIX API
    - Perhaps wrappers to manage complexity of exception handling
      (*e.g.*, Bell Labs XEPT work)
Next Step: Robustness Testing Service

- **Ballista Server**
  - Selects tests
  - Performs pattern Analysis
  - Generates “bug reports”
  - Never sees user’s code

- **Ballista Client**
  - Links to user’s SW under test
  - Can “teach” new data types to server (definition language)
Wrap-up

◆ “Lofty Goal:” harden legacy and COTS software components
  • For mission-critical systems
    Without extensive re-engineering to improve robustness

◆ Robustness metric for Operating Systems
  • Failure rates look high; true impact depends on operational profile
  • Controversy as to whether Abort failures are OK
  • Metrics help stimulate demand for improvement

◆ Ballista robustness testing approach
  • Scalable, portable, reproducible
  • C library has higher failure rate, less diverse than OS system calls
  • Currently available as web server; applying to several domains

◆ Future: Windows NT, more system state, heavy system loads
http://www.ices.cmu.edu/ballista