Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs

Phil Gibbons

15-712 F15

Lecture 6

Today's Reminders

- **Kevin office hours**
  - 2-4 pm Tues @ CIC 4th floor

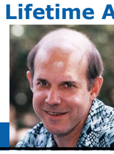
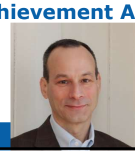
Eraser: A Dynamic Data Race Detector for Multi-Threaded Programsm

Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas E. Anderson [SOSP'97]

- **Stefan Savage (UCSD, CMU undergrad, ACM Fellow)**
- **Michael Burrows (Google, BWT in bzip2, FRS Fellow)**
- **Greg Nelson (HP, d. 2015, Herbrand Award 2013)**
- **Patrick Sobalvarro (Upward Labs, many start-ups)**
- **Tom Anderson (U. Washington, 35000+ citations, Usenix Lifetime Achievement Award 2014)**

Best Paper



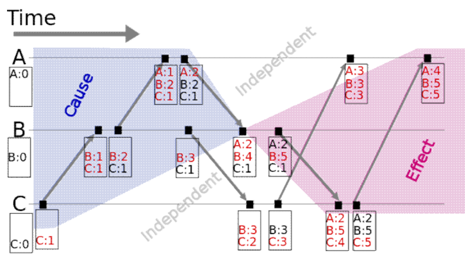




Data Race Detection

- **Data Race: Two concurrent threads access a shared variable and**
  - At least one access is a write
  - The threads use no explicit mechanism to prevent the accesses from being simultaneous
- **Monitors prevent data races, but only when all shared variables are static globals**
- **Static Analysis must reason about program semantics**
- **Happens-before Analysis**
  - E.g., using vector clocks
- **This paper: based on locking discipline**

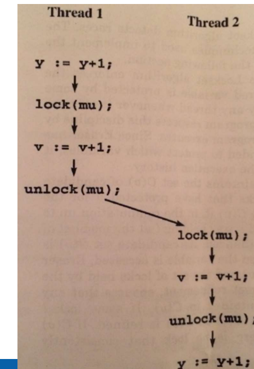
Vector Clocks for Race Detectors

- $a \rightarrow b$ **iff** $V(a) < V(b)$
- **Use vector clocks**
  - Inter-thread arcs are from unlock L to next lock L; otherwise, we report a data race
  - Check each access for conflicting access unrelated by →



Drawbacks of Happens-Before

- **Difficult to implement efficiently**
  - Require per-thread info about concurrent accesses to each shared-memory location
  - Effectiveness highly dependent on interleaving that occurred:



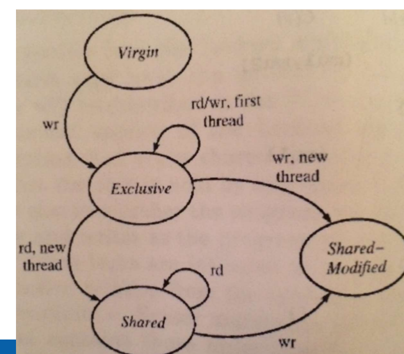
Lockset Algorithm (1st version)

- **Let locks_held(t) be the set of locks held by thread t**
- **For each v, initialize C(v) to the set of all locks**
- **On each access to v by thread t:**
  - Set C(v) := C(v) ∩ locks_held(t)
  - If C(v) is empty, then issue a warning

Program	*locks_held*	*C(v)*
	{}	{mu1,mu2}
lock(mu1);		
	{mu1}	
v := v+1;		
		{mu1}
unlock(mu1);		
	{}	
lock(mu2);		
	{mu2}	
v := v+1;		
		{}
unlock(mu2);		
	{}	

Handling Initialization & Read Sharing

- **State machine tracked for each variable v**
- **Empty Lockset C(v) reported only if v is Shared-Modified**



Lockset Algorithm (Final Version)

- **Let locks_held(t) be the set of locks held by thread t;**
  Let write_locks_held(t) be set of locks held in write mode
- **For each v, initialize C(v) to the set of all locks**
- **On each read of v by thread t:**
  - Set C(v) := C(v) ∩ locks_held(t)
  - If C(v) is empty, then issue a warning
- **On each write of v by thread t:**
  - Set C(v) := C(v) ∩ write_locks_held(t)
  - If C(v) is empty, then issue a warning

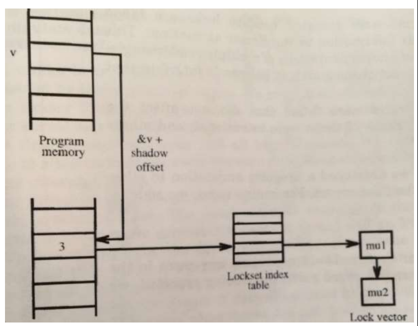
Locks held purely in read mode do not protect against a data race between the writer & some other reader thread

Implementation

- **Binary instrumentation**
- **Instruments each load/store and malloc to maintain C(v)**
  - 32-bit (aligned) words
  - But not stack-based accesses (stack assumed private)
- **Instrument lock/unlock calls, thread init/finalize to maintain lock_held(t);**
- **Warnings report file, line number, thread ID, memory access address & type, PC, SP**
  - Option: Log all accesses to v that modify C(v)

Representing C(v)s

- **Represent by small integer lockset index into table**
  - Never observed > 10K distinct lock sets
- **Append-only table**
- **Lock vectors sorted**
- **Cache results of set intersections**
- **Shadow word: 30-bit index, 2-bit state**



- **Issue: Shadow memory doubles size of memory**

Performance

"Performance was not a major goal in our implementation"

- **Typical app slowdown: 10x-30x**
  - Estimate half due to procedure call at every load/store
  - Today: dynamic binary instrumentation (DBI) using inlining for short code segments

"Eraser is fast enough to debug most programs and therefore meets the most essential performance criterion."

Source of Overheads

- **We measured 55x-70x for Valgrind 2.2 [2006]**
  - Lockset work overheads: 7x-10x
  - Instrumentation overheads: 48x-60x
    - Compete for cycles & resources (register state, L1 cache)
    - Must recreate HW state (effective addresses, IP)
    - e.g., 3 x86 insts becomes 27 x86 insts:

```
c = a + b;
```

(a) Original C statement

```
mov 0xffffffe8(%ebp),%eax
add 0xffffffec(%ebp),%eax
mov %eax,0xffffffe4(%ebp)
```

(b) Original x86 assembly code

```
LEA1L -24(%ebx), %eax                 # Determine addr of first read
    leal 0xFFFFFFE8(%ebx), %eax
CCALLo 0xB113C900(%eax)               # Call read_check()
    pushl %eax                        # Save register
    call * 36(%ebp)
    popl %eax                         # Restore register
LDL (%eax), %ecx                      # Do the actual read
    movl (%eax), %ecx
INCEIPo $3                            # Update the instrumented EIP
    movb $0x5D, 0x44(%ebp)
LEA1L -20(%ebx), %eax                 # Determine addr of second read
                 ⋮
```

(c) x86 assembly code instrumented for ADDRCHECK

Figure 3: Illustrating the sources of overheads in dynamic binary instrumentation.

False Alarms & Annotations

- **Memory reused without resetting shadow memory**
  - When app uses private memory allocator
  - Annotation: EraserReuse(address, size)
- **Synchronization outside of instrumented channels**
  - Private lock implementations of MR/SW locks
  - Spin on flag
  - Annotation: EraserReadLock(lock), EraserReadUnlock(lock),
    EraserWriteLock(lock), EraserWriteUnlock(lock)
- **Benign races**
  - Annotation: EraserIgnoreOn(), EraserIgnoreOff()

"We have found that a handful of these annotations usually suffices to eliminate all false alarms."

Race Detection in OS Kernel

- **OS often raises the processor interrupt level to provide mutual exclusion**
  - Particular interrupt level inclusively protects all data protected by lower interrupt levels
  - Solution: Have a virtual lock for each level; when raise level to x, treat this as first x per-level locks acquired
- **OS makes greater use of POST/WAIT style synch, e.g., semaphores to signal when a device op is done**
  - Problem: Hard to infer which data a semaphore is protecting

Experience

- **Ten iterations to resolve all reported races**
- **Worked well on servers: Evidence that experienced programmers tend to obey the simple locking discipline**
- **AltaVista Web indexing service: mhttpd & Ni2**
  - Some good examples of benign races in production codes
  - 24 annotations reduced false positives from 100+ to 0
  - Reintroduced two old bugs & found/corrected in 30 minutes
- **Vesta Cache Server**
  - Found data race on "valid" bit—serious on weak memory model
  - Benign: Main thread passes RPC request to worker thread;
    Head of log lock makes entire log private
  - 10 annotations & 1 bug fix reduced alarms from 100s to 0

Experience

- **Petal distributed storage system**
  - Implements distributed consensus, failure detector/recovery
  - Found one real race
- **Undergraduate coursework**
  - 100 runnable assignments
  - Found data races in 10% of them
- **Sensitivity to thread interleavings**
  - Reran Ni2 & Vesta on 2 threads instead of 10
  - Same race reports, in different order

Protection by Multiple Locks

- **Every writer must hold all locks**
  Every reader must hold at least 1 lock
  - Used to avoid deadlock in program that contains upcalls
- **Causes false alarms**
  - Not worth cost of handling this

Deadlock

"If the data race is Scylla, the deadlock is Charybdis."

(Sea monsters in Homer's Odyssey)

- **Discipline: Acquire locks in ascending order**
- **Found cycle of locks in formsedit application**
- **Would be useful addition to Eraser…**

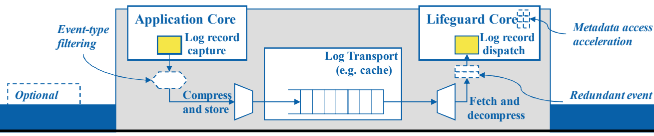
Bugs as Deviant Behavior

Dawson Engler, David Chen, Seth Hallem, Andy Chou, Benjamin Chelf [SOSP'01]

- **Infer programmer beliefs from source code**
  - E.g., <a> must be paired with <b>
- **Cross-check for contradictions**
- **Report in order of likelihood of belief accuracy**
- **Developed 6 template checkers that found 100s of bugs in real systems such as Linux and OpenBSD**

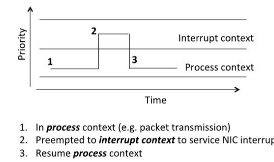
Data Race Detection Today

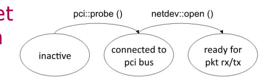
- **Valgrind tools: Helgrind, DRD, Tsan**
  - Use Happens-before; Only Tsan also uses Lockset
  - Early versions of Helgrind used Lockset
- **Intel ThreadChecker: uses Happens-before**
- **Papers proposing hardware support, e.g.,**
  - HARD [HPCA'07]: HW bloom filters for fast lockset ops
  - LBA [ISCA'08]: HW to eliminate instrumentation overheads, run analysis tool on different core + Idempotent Filters: Caches recent addresses, ignores accesses that hit in cache, flushes on lock/unlock; Overheads down to 1.4x (2x the cores)



Data Races in Kernels

- **DataCollider [OSDI'10]**
  - Stalls a kernel thread in critical sections to see if racy access occurs while stalled (not for time-critical interrupts)
- **Guardrail [ASPLOS'14] for kernel-mode drivers addresses the following challenges:**
  - Single thread can race itself (!)
  - Synchronization invariants based on context of device state
  - Synchronization based on deferred execution using softirqs or timers
  - Mutual exclusion via HW test-and-set or disabling interrupts & preemption





Data Races in Parallel Codes

- **Cilk: Nondeterminator, Cilkscreen**
  - Relies on fork-join structure of Cilk programs to determine whether two conflicting accesses are ordered
  - Reports race or that no race can occur with the given input
  - Runs serially
- **Parallel detectors for parallel code**
  - Issue: Capture & enforce in analysis the app's inter-thread data dependencies
  - Issue: Metadata access atomicity, especially under weak memory models
  - E.g., Paralog [ASPLOS'10], Butterfly Analysis [ASPLOS'10, PACT'12, PACT'15]

Wednesday's Paper

Using Model Checking to Find Serious File System Errors

Junfeng Yang, Paul Twokey, Dawson Engler, Madanlal Musuvathi [OSDI'04]