Automatic Race Detection
Adam Purtee
CSC 456
November 18, 2010
Outline

- Race Detection Methods
  - Static vs Dynamic
  - Happens-before
  - Lock-sets
  - Sampling
- Kernel Specific Issues
- Race Detection Tools
- RELAY
  - Overview
  - Almost Sound
  - Relative Locksets
  - Results
- DataCollider
  - Overview
  - Sampling
  - Conflict Detection
  - Results
- Summary
Race Detection: Methods

- **Static Detection**
  - May not scale well
  - Excessive false positives – leads to missing actual races, or introducing bugs to fix nonexistent races

- **Dynamic Detection**
  - Excessive overhead
  - Errors frequently go undetected
Race Detection Methods

- Happens-before
  - Described by Leslie Lamport in 1978
  - Often implemented using vector clocks (adds linear space/time overhead in number of threads)

- Lock-sets
  - Requires thorough understanding of synchronization methods
  - Successfully used in a number of systems (both static and dynamic)

- Sampling
  - The primary method described in this talk
  - A newer detection technique
Detection via happens-before relations

Happens before can be implemented via vector clocks, which are arrays of \( N \) timestamps, where \( N \) is the number of threads of execution. The clocks are maintained according to the following rules:

- The clocks begin at zero
- Every process increments its clock at every operation
- All messages include the clock vectors
- Upon receiving a vector, a process updates its own vector if the received value is greater for any clock
Detection via lock-sets

Keep track of all locks held for each writable memory location, issue a warning if a shared memory object is not protected by any locks while modified.

The original (simplest) lock-set algorithm (from Eraser):
Let $v$ be a shared variable and $C(v)$ be a set of “candidate” locks. Let $\text{locks\_held}(t)$ be the sets of locks held by thread $t$.
For each $v$, initialize $C(v)$ to be the set of all locks.
On each access to $v$ by thread $t$, set $C(v) = C(v) \cap \text{locks\_held}(t)$.
If $C(v) = \{\}$, then issue a warning.
Lock-sets: Example (from Eraser)

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

Kernel Specific Issues

- General code size/complexity:
  - Linux: 4.5M lines (2.4 kernel)
  - Windows 7: unknown, but likely several million lines
- Complex synchronization mechanisms
- A drastically slowed down Kernel leads to an (essentially) unrunnable OS.
Kernel Race Detection Tools

- RELAY
- DataCollider
- Many non-kernel race detectors available online: LightRace, PACER, Eraser, RacerX, etc.  
RELAY: Overview

- Static race checker used on the Linux kernel
- Almost sound (4 sources of errors)
- Uses relative lock-sets (as opposed to absolute/global lock-sets)
- Can be ran highly parallel

An analysis of 4.5 million lines of C (Linux Kernel) took 5 hours on the UCSD FWGrid cluster and detected a total of 53 live races.
RELAY: Almost Sound

- Ignores blocks of included assembly code
- Boundaries of multidimensional arrays
- Function pointers
- Excessive pruning of warnings
Instead of maintaining a list of every lock held by every thread, a relative lockset is computed on a per-function basis. It is the set of all changes in locks held through function execution. RELAY reports a warning when two accesses are to the same location, at least one access is a write, and the intersection of their positive locksets is empty.

This type of information makes RELAY scalable. For example (from the RELAY paper): Variable f may be accessed inside function x while holding all locks that were held on entry, plus x.lock1, and minus x.lock2.
RELAY: Sample Detected Race

```c
iounmap(volatile *addr) {
    read_lock(&vmlist_lock);
    for (p = vmlist; p; p = p->next) {
        if (p->addr == addr) break;
    }
    read_unlock(&vmlist_lock);
    change_page_attr(virt_to_p(p->phys_addr),
        p->size >> PAGE_SHIFT);
}

/* called with write_lock(vmlist_lock) */
__remove_vm_area(*addr) {
    for (t = vmlist; t != NULL; t = t->next) {
        if (t->addr == addr) break;
    }
    unmap_vm_area(t);
    t->size -= PAGE_SIZE;
    return t;
}
```
DataCollider: Overview

- Dynamic race checker developed for the Windows 7 kernel
- Oblivious to synchronization methods
- Provides stack traces / core dumps
- Interesting sampling mechanism
- Conflict detection
- Avoids false positives
- Pruning of begin results
- Empirical results
DataCollider: Sampling

- Disassemble and generate sampling set (locations that access memory)
- Remove accesses to volatile locations, to prune synchronization ops
- Sampling locations eventually cluster to infrequently executed code
- Must sample both accesses
- Static code sampling (equal preference to rarely executed code)
- Soft break points
- Hard break points (read/write, x86 supports four)
- Controlling sampling rate (dynamic adjustment to hit target rate)
DataCollider: Conflict Detection

- Delaying threads (up to 15ms, depending on IRQL)
- Data breakpoints
  - Hardware traps (write r/w, read write)
  - Uses interrupts to update all processor breakpoints
  - Can break on 1B, 2B, or 4B locations (up to four)
  - When a trap fires then a race has been detected (and both threads identified)
  - Multiple address spaces and >16B shared memory handled via repeated reads
- Repeated-reads
  - Repeatedly read from a location and watch for changes
  - Only detects one of the racy threads
DataCollider: Results

- Implemented for 32-bit Windows 7 Kernel
- Detected 25 races with overhead less than 5%
- Tested on dual quad-core processor machine
void AddToCache() {
    // ...
    A: x &= ~(FLAG_NOT_DELETED);
    B: x |= FLAG_CACHED;

    MemoryBarrier();
    // ...
}

AddToCache();
assert( x & FLAG_CACHED);

The cached bit might not get set if x is modified between A and B.
Summary

- Race detection Methods
- Challenges detecting races in kernels
- RACER
- DataCollider
References

- Effective Data-Race Detection for the Kernel (OSDI ’10)
  Erickson, Musavathi, Burckhardt, and Olynyk.

- RELAY: static race detection on millions of lines of code
  (SIGSOFT ’07)
  Young, Jhala, and Lerner.
  http://cseweb.ucsd.edu/~lerner/papers/relay.pdf

- Data Race Detector Links:

- Eraser: A Dynamic Data Race Detector for Multithreaded Programs
  Savage, Burrows, Nelson, Sobalvarro, and Anderson.
  www.stanford.edu/class/cs240/readings/eraser.pdf
Portions of the material used in these slides was taken from the papers listed in the references (especially the examples and algorithm descriptions). These slides are to be used only for the presentation in CS486 as a summary of what was learned through reading those papers.