Eraser

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

S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, T. Anderson

Serge Koren

CMSC714

Outline

- Introduction
- Related approaches
- Eraser's approach
- Implementation/Experience
- Conclusion

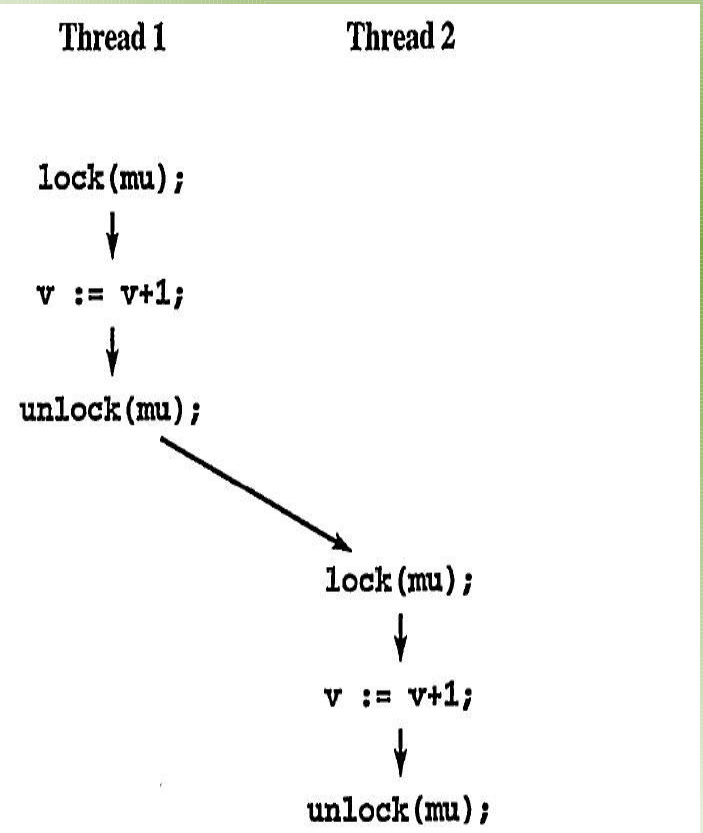
Introduction

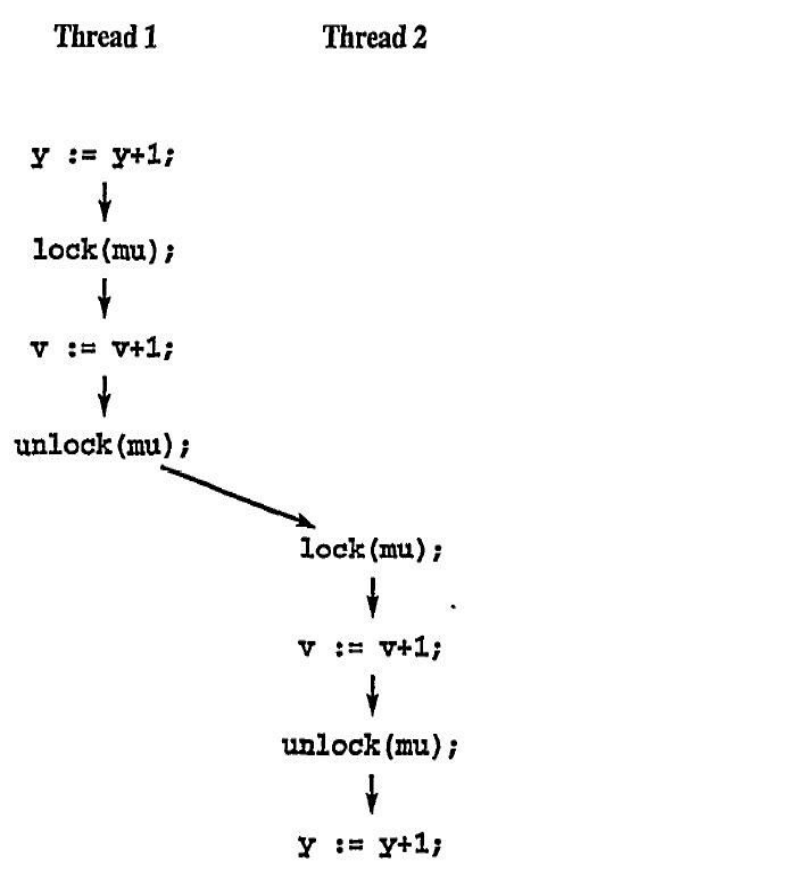
- Threaded programming is everywhere
- Hard to debug, “Threads are a bad idea” (Osterhout 96)
- Data Race
  - Two concurrent threads access shared variable
    - At least one access is a write
    - No mechanism to prevent accesses from being concurrent

Related Approaches

- Monitors – state and procedures together
- *Happens-before*
  - Events ordered between threads based on synchronization objects they access
  - More general, more costly
  - Less thorough

Happens-before





Lockset Algorithm

- Support only lock-based synchronization
- Check all read/writes as program runs
  - Infer protection from execution history

- On each access to variable, v, by thread, t
  - Set $C(v) = C(v) \cap$ *locks_held(t)*
  - If $C(v) = \{\}$, issue warning

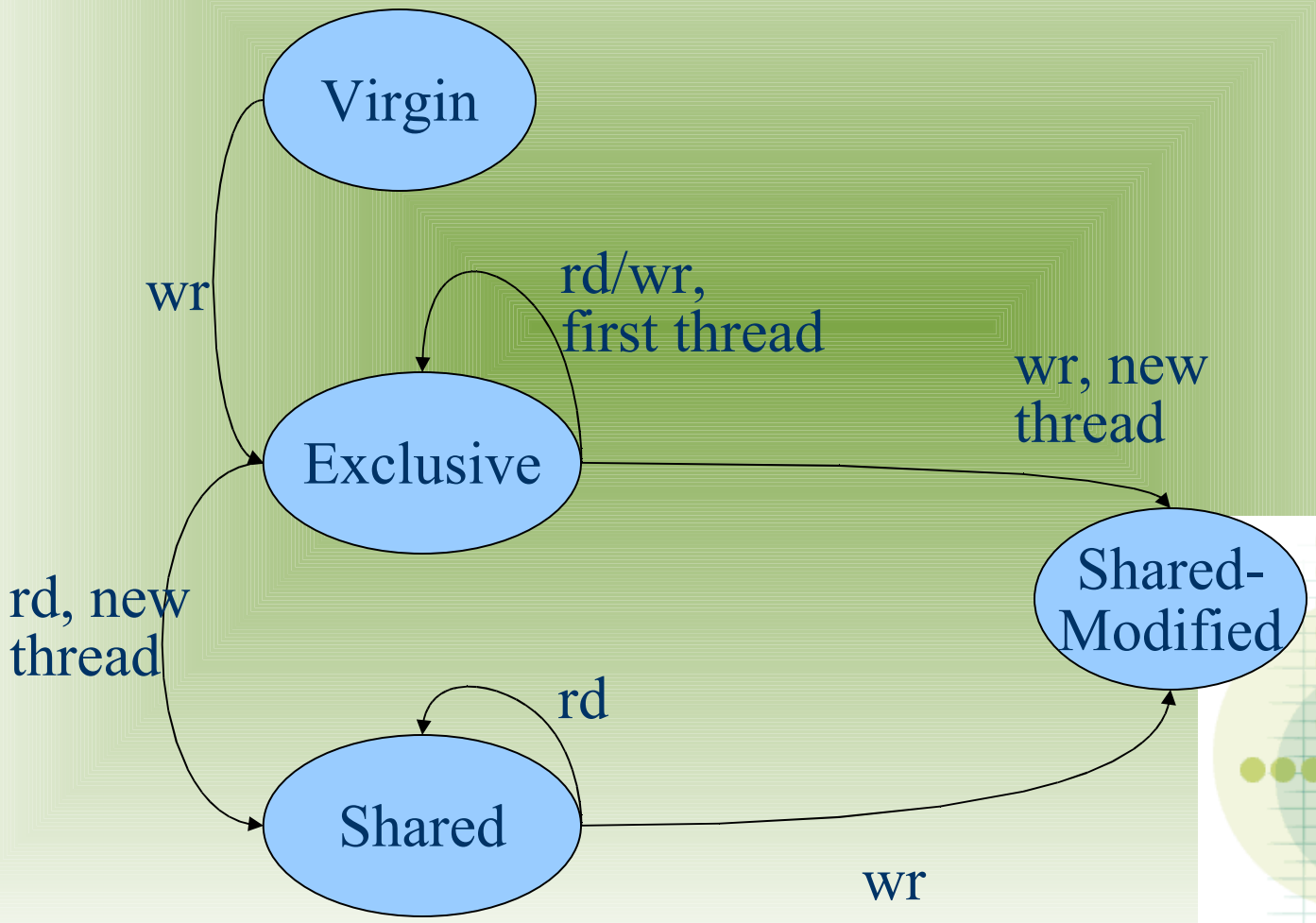
Example

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

Improving Lockset

- Initialization – initialize a shared variable without holding lock
- Read-shared data – written only at initialization and then read (no need for locks)
- Read-write locks – read by multiple threads, written by only one at a time

Improved Lockset Example



Read-Write Locks

- Lock m *protects* v if m held in write for writes and in read/write for read
- On each read of variable, v, by thread, t
  - Set C(v) = C(v) ∩ *locks_held(t)*
  - If C(v) = {}, issue warning
- On each write of variable, v, by thread, t
  - Set C(v) = C(v) ∩ write_*locks_held(t)*
  - If C(v) = {}, issue warning

Implementation

- Instrument program
- Indexed table of lock sets (hashed and sorted)
- Shadow word
  - Exists for every 32-bit data segment
  - State condition
  - Lockset index
- 10-30 times slower than without

Annotations

- Memory reuse – private allocators/free lists
  - EraserReuse
- Private locks – private implementations of locks
  - EraserRead/WriteLock/Unlock
- Benign races – race that does not affect correctness
  - EraserIgnoreOn/Off

Experience

- Tested on different code bases
  - Found true data races, found false alarms
- Effectiveness/Sensitivity
  - Found data races that had existed before
  - Works the same on two or ten threads
- Multiple locks
  - Many read locks, write takes all read locks
- Deadlock detection using ordering of locks

Conclusion

- Little data on sensitivity
- Only works for locks (pthreads)
- Annotations needed
- Slow/doubled memory usage
- Multiple lock fix incorrect/unclear

- http://valgrind.org (Helgrind: a data-race detector)