TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

Effective data-race detection for the kernel

J. Erickson, M. Musuvathi, S. Burckhard, K. Olynyk

presented by Bjoern Doebel

Data races

- Common definition:
  - n >= 2 threads access a memory location concurrently
  - At least one access is a write.
  - No explicit mechanism to prevent simultaneous access is used.

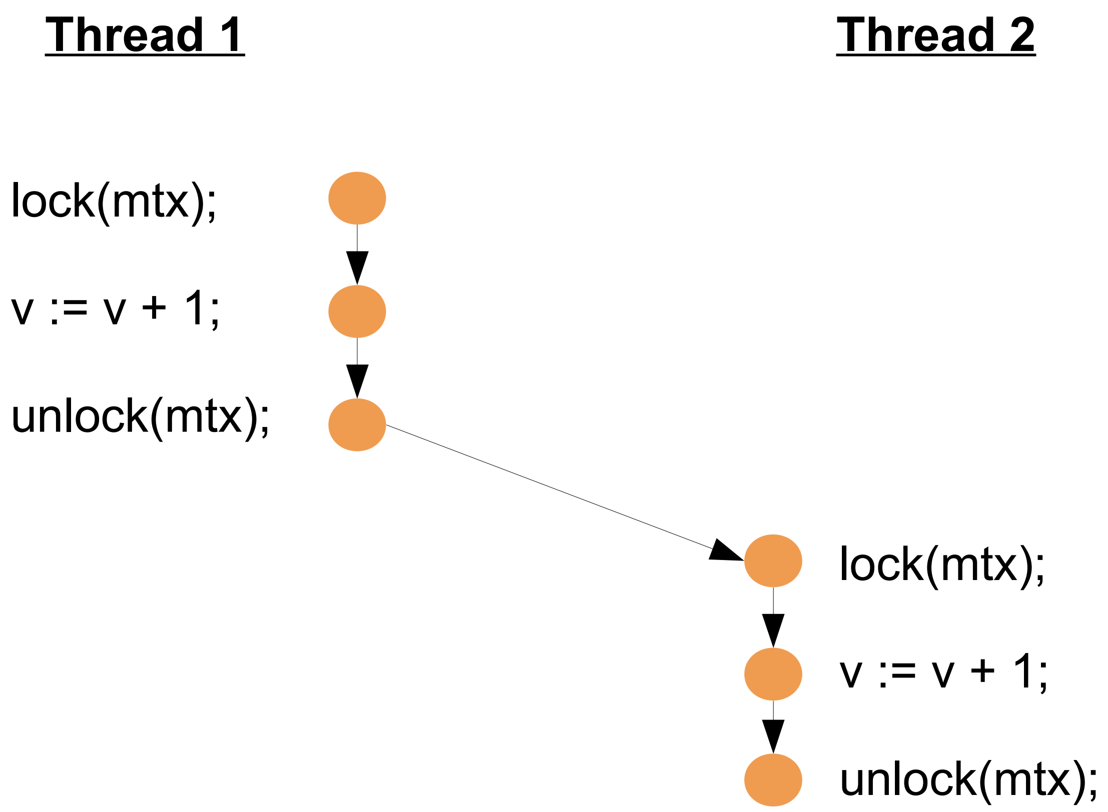
- Happens a lot
  - → many uncritical errors → *benign data race*
    - Update of statistical values
    - Spinlocks & Co. → *ad-hoc synchronization*
    - only know with thorough understanding of the code
  - Sync. primitives have multiple uses

- Reproducing a race is tricky.

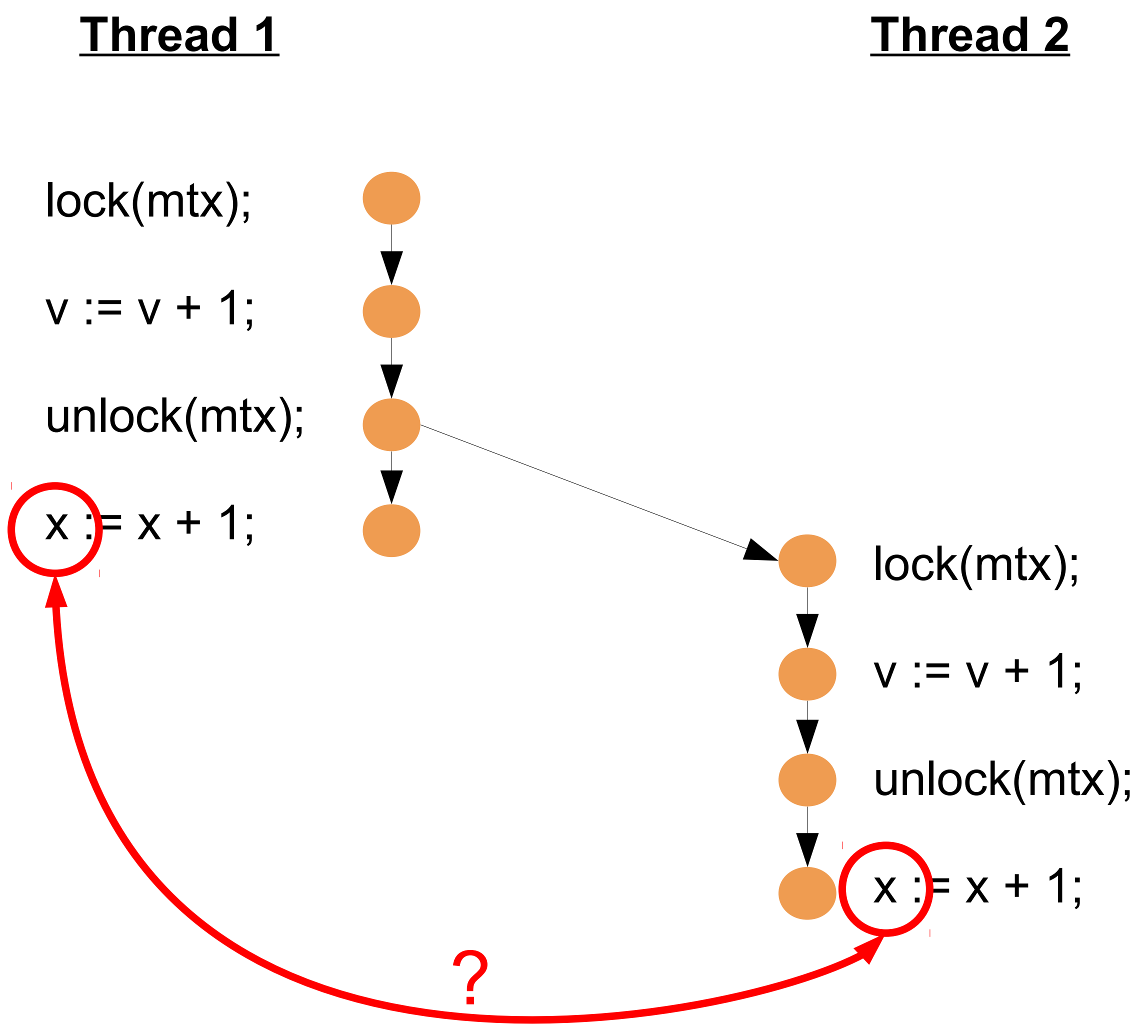
Race-detection: Happens-before relations

- Obtain trace of
  - Memory accesses
  - Use of locking primitives
  - [Netzer1991]: use of MPI messaging primitives
- Order executed instructions in happens-before relation:
  - Sequentially executed instructions in one thread
  - unlock()/lock() on a mutex implies inter-thread happens-before
- Memory accesses are flagged as races, if no happens-before relation can be established
- [Lamport 1978] [Netzer 1991,1993]

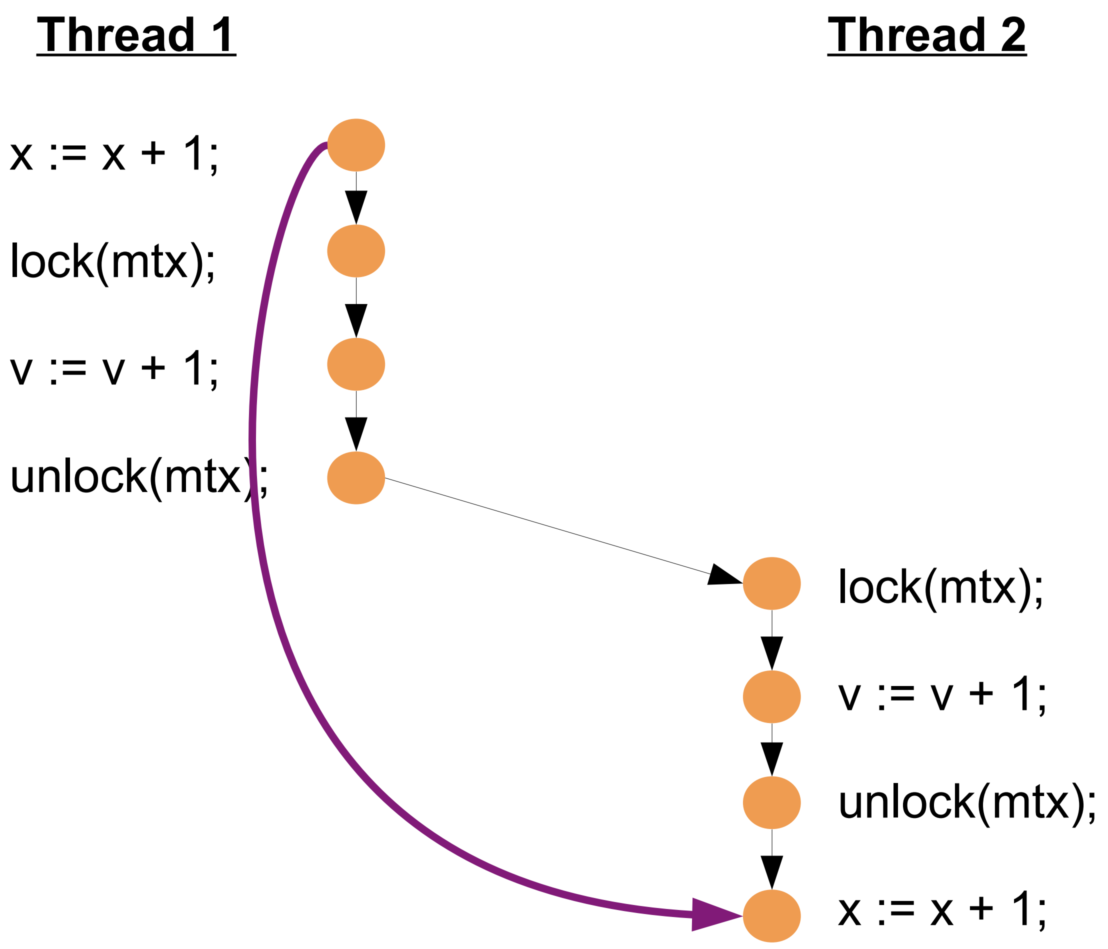
Race-detection: Happens-before relations



Race-detection: Happens-before relations



Happens-before going wrong...



Race detection: Lockset analysis

- Monitor locking primitives only

- Let LOCKS(t) be the set of locks held by thread t
- For each value V initialize C(V) to the set of all locks
- For each memory access to V by t:

$$C(V) := C(V) \cap LOCKS(t)$$

$$Error\ if\ C(V) = \emptyset$$

- Dynamic [Savage 1997] and static [Engler 2003] versions available

Race detection: Classification

- Common race detectors use either HBR or LSA or a hybrid approach
- Next step: tell apart benign from critical races
- Automation using record/replay analysis
  - Record/replay makes reproduction trivial.
  - Classification:
    - Try out all possible schedules in replay
    - Compare states after a certain point
  - Binary-level [Naraynasamy 2007]
    vs. language-level [Shen 2008]
  - Add optimizations to determine which schedules are interesting [Musuvathi 2008]

Race detection and the OS

- Threads may execute in different contexts → no clean abstraction w.r.t. data races
  - Racy accesses may be observed in the same thread
- Many more synchronization primitives, e.g.
  - Spinlocks
  - CLI/STI
  - Semaphores
- Accesses to/from device memory
  - External state changes modify memory content
  - DMA
- Must not have unacceptable overhead

- Preprocessing: generate a set M of all memory accesses of a program
- Periodically:
  - Pick k random elements from M and set instruction breakpoints
- On instruction breakpoint:
  - Perform conflict detection
  - Randomly pick another element from M and set an IBP
- Post-processing
  - Database of situations known to be benign races

Conflict detection

- Alternative A
  - Set data breakpoint (r or rw) on memory location
  - Delay execution
  - If breakpoint hits: ERROR
  - Issues
    - Doesn't work for device memory
    - Limited # of breakpoint registers
    - Miss: virtual address mapped to same physical address
    - False pos: same virtual address in different address spaces
- Alternative B
  - Read value of location
  - Delay execution
  - Read value again
  - On mismatch: ERROR
  - Issue: Only one of the race participants is known

Discussion

- Still facing the major problem of dynamic analysis: only works on the values actually observed.
- The real cool engineering is in the pruning database for benign races, which they don't talk about at all.
- Main focus still is on code that is executed often.
- *"But these techniques still suffer from the cost of sampling, performed at every memory access. DataCollider avoids this problem by using hardware breakpoint mechanisms."* → That's not the whole story!
- Removing thread-local stack operations from check set → may miss weird stuff such as DMA to stack?
- Fishy performance evaluation is fishy.