

User-level Mutual Exclusion



Lock-free?

- Avoid needing locking by using lock-free data structure
 - Still need short atomic sequences
 - compare-and-swap
- Lock-based data structure also need mutual exclusion to implement the lock primitive themselves.



How do we provide efficient mutual exclusion to kernel-implemented threads at user-level

- Interrupt disabling?
- Syscalls?
- Processor Instructions?



Optimistic Approach

- Assume the critical code runs atomically
 - *Atomic Sequence*
- If an interrupt occurs, OS recovers such that atomicity is preserved
- Two basic mechanisms
 - Rollback
 - Only single memory location update
 - Guarantee progress???
 - Rollforward



How does the OS know what is an atomic sequence?

- Designated sequences
 - Match well know sequences surrounding PC
 - Matching takes time
 - sequence may occur outside an atomic sequences
 - Rollback might break code
 - Rollforward okay
 - Sequences can be inlined
 - No overhead added to each sequence, overhead only on interruption



- Static Registration
 - All sequences are registered at program startup
 - No direct overhead to sequences themselves
 - Limited number of sequences
 - Reasonable to identify on interrupt
 - No inlining

- Dynamic Registration
 - Share a variable between kernel and user-level, set it while in an atomic sequence
 - Can inline, even synthesize sequences at runtime
 - Adds direct overhead to each sequence



How to roll forward?

- Code re-writing
 - Re-write instruction after sequence to call back to interrupt handler
 - Cache issues

- Cloning
 - Two copies of each sequence
 - normal copy
 - modified copy that call back into interrupt handler
 - On interrupt, map PC in normal sequence into PC in modified
 - Mapping can be time consuming
 - Inlining???



- Computed Jump
 - Every sequence uses a computed jump at the end
 - Normal sequence simply jmp to next instruction
 - Interrupted sequence jumps to interrupt handler
 - Adds a jump to every sequence

- Controlled fault
 - Dummy instruction at end of each sequences
 - NOP for normal case
 - Fault for interrupt case
 - Example is read from (in)accessible page
 - Good for user-kernel privilege changes
 - Still adds an extra instruction

Limiting Duration of ROLLforward

- Watchdog
- Restriction on code so termination can be inspected for



