LECTURE - 27

Lecture Outline

- Synchronization mechanisms, and consistency models
-
- ***Scribe for today?***

Synchronization

- Required since communication is through shared memory
- Synchronization primitives
  - Involve atomic read-and-write of a memory location
  - Atomic exchange (with a register)
  - Test-and-set
  - Fetch-and-increment

Synchronization and Coherence

- Atomic read-and-write causes problems with coherence
  - Additional complexity
- Solution: push complexity to software!
  - Pair of instructions
  - Hardware support to tell if the two were executed atomically
  - *Load-linked* and *store-conditional*
  - Store fails if any intervening process switch, or coherence control operation

Load-Linked/Store-Conditional

- Can implement atomic-exchange, fetch-and-increment
- Implementation issues:
  - Use a link register to store address in previous load-linked instruction
  - Process switch, or coherence control operation will clear the link register
  - Store-conditional succeeds iff the address in it matches that in the link register
- Beware of what (and how many) instructions between the pair

Using Atomic Exchange for Spin-Locks

- Processor spins until it gets access to lock
- Useful to test if lock is already held, before trying to lock
- Even then, performance problems when multiple processors are trying to grab the lock
  - Read/write misses generated by all processors
  - Misses satisfied sequentially

Barrier Locks

- Barrier is a synchronization primitive
  - Can be used in programs
  - Forces all processors to wait until the last one reaches the barrier
- Can be implemented with two spin-locks
  - One to increment a counter
  - One to hold the processors until barrier
- Can cause deadlock!
  - Use count-down, or sense-reversing barrier

Performance Optimizations

- Exponential back-off
- Queuing locks