Lock-basics

- Why necessary? - One form of synchronization primitive required to provide mutual exclusion property across critical sections
- What is lock? Lock is a data structure that could be in various lock states (similar to process abstraction and its states)
  - LOCKED: No one else is allowed to enter the critical section when the lock is in LOCKED STATE. The thread holding the lock is the owner of the lock and there could be only one owner for a lock (constraint on ownership is not true for complex form of locks like reader writer locks)
  - UNLOCKED: Any one thread could grab the lock and access the critical section. There should be no owner when lock is in this state.

Building locks

- Simple lock: Context switch while in critical section could lead to erroneous behavior and thus disabling interrupts could prevent context switches. This could serve as a locking mechanism but this does not work on multi-core cpus or applications could be relied on to disable and enable interrupts properly.
- The primary requirement for building a lock is to change the lock state from UNLOCKED to LOCKED or vice versa atomically. Mutual exclusion cannot be guaranteed without this property.
- Hardware primitives are leveraged to provide such atomic guarantees (but completely software based locks have been built) and locks were built using them
  - TestAndSet (XCHG instruction in x86)
  - CompareAndSwap (CMPXCHG instruction in x86)
  - LoadLinked and Store conditional (Available in PowerPC architecture)

Spinlocks

- Keep checking the lock state till it changes to UNLOCKED state.
- If there are N threads waiting to grab the lock, it is not clear which thread would get hold of the lock. The first thread that started waiting for the lock might not get the lock when it gets released. Fairness property is not provided by spinlocks.

Ticket locks

- New lock state in addition to LOCKED and UNLOCKED. Every lock has a turn number.
- Every waiting thread gets a ticket number and only when the lock's turn number is equal to the waiting thread's ticket number, the thread gets to run.
- Since the waiting threads get a ticket number in the order in which they started waiting, they get the lock in the same order. Thus fairness is guaranteed.
- This is a form of spin lock using FetchAndAdd primitive (XADDL instruction in x86)

Spinlocks waste time

- Continuous checking of lock states results in wasting CPU cycles.
In a single core system, this could result in wasting the waiting thread's schedule quantum and also delaying the lock owner from getting execution. This in turn delays the waiting thread.

This situation is reduced to some extent in multi-core CPUs but not completely eliminated. If number of waiting threads are greater than number of CPUs then the same problem explained in above point could happen.

Performance property of the lock is not good for spinlocks. However, spinlocks are good for smaller critical sections.

Yield instead of spinning

- If spinning is bad, then the waiting thread could give back (yield) the CPU and does not waste spinning during its schedule quantum.
- Queued locks are used to provide both fairness and performance.
  - Waiting threads are pushed into a waiting queue associated with the lock structure.
  - When the lock gets released, a thread from the waiting queue is woken up and is allowed to become owner of the lock.
  - Fairness: Since the lock is given to threads in the order in which it entered the waiting queue, fairness is guaranteed.
  - Performance: Since the waiting threads just get queued rather than spinning, cpu cycles are not wasted.

Hybrid or two-phase locks

- Combining spinning and sleeping give rise to the hybrid approach
- Spinning is good for smaller critical sections. When a thread tries to grab a lock and if the lock is in LOCKED state, it spins for a shorter period of time and then enters in to the sleep state.
- The idea is that if the critical region is small, then the lock will be released soon and so, it is beneficial to spin rather than sleeping directly. Sleeping is beneficial only when the wait time is high else its costly too.
- Futex locks in Linux takes this hybrid approach.

Other form of locks

- Reader-writer locks
- RCU (Read copy update)
- Recursive locks
- per-cpu locks (e.g. similar to sloppy counters)