Mutex

结构变量设计 ：

```
type Mutex struct {
   state int32 //state 4  int,  goroutine
   sema  uint32 //ema  0 goroutine park sema acquire  sema  0sema release
sema  goroutine
}
type Locker interface {
   Lock()
   Unlock()
}




const (
   mutexLocked = 1 << iota // mutex is locked                1
   mutexWoken // 2   mutex is woken up             10
   mutexStarving // 4  mutex is starving         100
   mutexWaiterShift = iota  //3               3
   starvationThresholdNs = 1e6 //1000000
)
```

一 ： 互斥锁最简单的一种实现：

Lock 流程 ：

```
1. state  0CAS
2.for
  +  1
  + CASstateforlock   waiter+1
```

lock代码：

```
if atomic.CompareAndSwapInt32(&m.state, 0, mutexLocked) {
   return
}
awoke := false //  goroutine  true
for {

   old := m.state //  m.state
   // todo
   new := old | mutexLocked //  mutexLocked  1
   //todo   old mutexLocked  0 state  waiter  +1
   if old&mutexLocked != 0 {
      new = old + 1 << mutexWaiterShift
   }
   if awoke {
      // The goroutine has been woken from sleep,
      // so we need to reset the flag in either case.  awoke  true,
mutexWoken  0
      new &^= mutexWoken
   }
   // CAS  m.state  old for
   if atomic.CompareAndSwapInt32(&m.state, old, new) {
      if old&mutexLocked == 0 { //  old mutexLocked  1 CAS  waiter  0
break
         break
      }
      runtime_Semacquire(&m.sema) //  sema <= 0  park
      awoke = true  //  goroutine  runtime  awoke true
   }
}
```

unlock 流程：

```
1. CAS
2. for next one  for  CAS
 waiter  0 goroutine old&(mutexLocked|mutexWoken) != 0  woken  goroutine
mutexLocked  for  old   CAS  new  woken  waiter  runtime_Semrelease
goroutine
```

unlock 代码实现：

```
new := atomic.AddInt32(&m.state, -mutexLocked)
if (new+mutexLocked)&mutexLocked == 0 { //  panic
   panic("sync: unlock of unlocked mutex")
}

old := new
for {//  state  waiter  0  return  old  mutexLocked|mutexWoken  1
goroutine
   // If there are no waiters or a goroutine has already
   // been woken or grabbed the lock, no need to wake anyone.
   if old>>mutexWaiterShift == 0 || old&(mutexLocked|mutexWoken) != 0 {
      return
   }
   // Grab the right to wake someone.  waiter  awoken
   new = (old - 1<<mutexWaiterShift) | mutexWoken
   if atomic.CompareAndSwapInt32(&m.state, old, new) {
      runtime_Semrelease(&m.sema) // cas  sema release  goroutine
      return
   }
   old = m.state
}
```

二： 互斥锁增加Spin 自旋

上边那种比较简单的实现的lock锁缺点 ：

```
sema  goroutine  FIFO
1.   goroutine Unlock  goroutine
2  spin  goroutine  park,  runtime
```

自旋相关的两个方法：

```
runtime_canSpingolangruntimeruntime_canSpin,
1. iter4
2. cpu1
3. 1
4. PPG

runtime_doSpinprocyieldPAUSEPAUSECPUPAUSECPU


//go:linkname sync_runtime_canSpin sync.runtime_canSpin
func sync_runtime_canSpin(i int) bool {
        if i >= active_spin || ncpu <= 1 || gomaxprocs <= int32(sched.
npidle+sched.nmspinning)+1 {
                return false
        }
        if p := getg().m.p.ptr(); !runqempty(p) {
                return false
        }
        return true
}


func sync_runtime_doSpin() {
        procyield(active_spin_cnt)
}
```

增加自旋的优化：

```
// Lock locks m.
// If the lock is already in use, the calling goroutine
// blocks until the mutex is available.
func (m *Mutex) Lock() {
    // Fast path: grab unlocked mutex.
    if atomic.CompareAndSwapInt32(&m.state, 0, mutexLocked) {
        if race.Enabled {
            race.Acquire(unsafe.Pointer(m))
        }
        return
    }

    awoke := false
    iter := 0
    for {
        old := m.state
        new := old | mutexLocked
        if old&mutexLocked != 0 { //
            if runtime_canSpin(iter) {
                // Active spinning makes sense.
                // Try to set mutexWoken flag to inform Unlock
                // to not wake other blocked goroutines.
                if !awoke && old&mutexWoken == 0 && old>>mutexWaiterShift !
= 0 &&
                    atomic.CompareAndSwapInt32(&m.state, old,
old|mutexWoken) {
                    awoke = true
                }
                runtime_doSpin()
                iter++
                continue
            }
            new = old + 1<<mutexWaiterShift
        }







        if awoke {
            // The goroutine has been woken from sleep,
            // so we need to reset the flag in either case.
            if new&mutexWoken == 0 {
                panic("sync: inconsistent mutex state")
            }
            new &^= mutexWoken
        }
        if atomic.CompareAndSwapInt32(&m.state, old, new) {
            if old&mutexLocked == 0 {
                break
            }
            runtime_Semacquire(&m.sema)
            awoke = true
            iter = 0
        }
    }

    if race.Enabled {
        race.Acquire(unsafe.Pointer(m))
    }
}
```

公平锁和非公平锁的实现：

```
mutex
1. starving
2. normal

1. normal -> starving : waiter1ms
2. starving -> normal : owner 1ms  -> switch
```