СБЕР

Внутреннее устройство мьютексов в Go

Сухов Илья
Главный эксперт

Мьютекс

это переменная, которая может находиться
в одном из двух состояний

Заблокирована

Не заблокирована

Виды мьютексов в Go

sync.Mutex

sync.RWMutex

Низкоуровневые примитивы

внутри мьютексов

Низкоуровневые примитивы внутри мьютексов

```
sync/atomic

func CompareAndSwapInt32(addr *int32, old, new int32) (swapped bool)
func AddInt32(addr *int32, delta int32) (new int32)
func LoadInt32(addr *int32) (val int32)
```

Низкоуровневые примитивы внутри мьютексов

```
runtime

func runtime_canSpin(i int) bool
func runtime_doSpin()
func runtime_nanotime() int64
func runtime_SemacquireMutex(s *uint32, lifo bool, skipframes int)
func runtime_Semrelease(s *uint32, handoff bool, skipframes int)
```

sync.Mutex

sync.Mutex

```go
type Mutex struct {
    state  uint32
    sema uint32
}

func (m *Mutex)Lock()
func (m *Mutex)TryLock() bool
func (m *Mutex)Unlock()
func (m *Mutex)lockSlow()
func (m *Mutex)unlockSlow(new int)
```

sync.Mutex

```go
const(
    mutexLocked = 1 << iota //  0b001
    mutexWoken // 0b010
    mutexStarving // 0b100
    mutexWaiterShift = iota // 3
    starvationThresholdNs = 1e6 //1_000_000 (1ms)
)
```

m.Lock()

```go
func (m *Mutex) Lock() {
    if atomic.CompareAndSwapInt32(&m.state, 0, mutexLocked) {
        return
    }
    m.lockSlow()
}
```

m.TryLock()

```go
func (m *Mutex)TryLock() bool {
    old := m.state
    if old&(mutexLocked|mutexStarving) != 0 {
    }
    if !atomic.CompareAndSwapInt32(&m.state, 0, mutexLocked) {
        return false
    }
    return true
}
```

m.Unlock()

```go
func (m *Muxtex) Unlock() {
    new := atomic.AddInt32(&m.state, -mutexLocked)
    if new != 0 {
        m.unlockSlow(new)
    }
}
```

m.lockSlow()

```go
func (m *Mutex) lockSlow() {
    var waitStartTime int64
    starving := false
    awoke := false
    iter := 0
    old := m.state
    for {
        ...
    }
}
```

sync.Mutex

m.lockSlow()

```
    for {
        if old&(mutexLocked|mutexStarving) == mutexLocked &&
runtime_canSpin(iter) {
            ...
        }
        ...
        if atomic.CompareAndSwapInt32(&m.state, old, new) {
            ...
        } else {
            old = m.state
        }
    }
```

m.lockSlow()

```
    if !awoke && old&mutexWoken == 0 && old >> mutexWaiterShift != 0 &&
atomic.CompareAndSwapInt32(&m.state, old, old|mutexWoken) {
        awoke = true
    }
    runtime_doSpin()
    iter++
    old = m.state
    continue
```

m.lockSlow()

```
    for {
        if old&(mutexLocked|mutexStarving) == mutexLocked &&
runtime_canSpin(iter) {
            ...
        }
        ...
        if atomic.CompareAndSwapInt32(&m.state, old, new) {
            ...
        } else {
            old = m.state
        }
    }
```

m.lockSlow()

```
new := old
if old&mutexStarving == 0 {
    new |=mutexLocked
}
if old&(mutexLocked|mutexStarving) != 0 {
    new += 1 <<mutexWaiterShift
}
if starving && old&mutexLocked != 0 {
    new |=mutexStarving
}
if awoke {
    if new&mutexWoken == 0 {
        throw("sync: inconsistent mutes state")
    }
    new &^=mutexWoken
}
```

sync.Mutex

m.lockSlow()

```
    for {
        if old&(mutexLocked|mutexStarving) == mutexLocked &&
runtime_canSpin(iter) {
            ...
        }
        ...
        if atomic.CompareAndSwapInt32(&m.state, old, new) {
            ...
        } else {
            old = m.state
        }
    }
```

sync.Mutex

m.lockSlow()

```
if old&(mutexLocked|mutexStarving) == 0 {
    break
}
queueLifo := waitStartTime != 0
if waitStartTime == 0 {
    waitStartTime = runtime_nanotime()
}
runtime_SemaquireMutex(&m.sema, queueLifo, 1)
starving = starving || runtime.nanotime() - waitStartTime >
starvationThresholdNs
old = m.state
if old & mutexStarving != 0 {
    ...
}
awoke = true
iter = 0
```

m.lockSlow()

```
if old&(mutexLocked|mutexWoken) != 0 || old >>mutexWaiterShift != 0 {
    throw("inconsistent mutex state")
}
delta := int32(mutexLocked - 1<<mutexWaiterShift)
if !starving || old>>mutexWaiterShift == 1 {
    delta -= mutexStarving
}
atomic.AddInt(&m.State, delta)
break
```

m.unlockSlow()

```
func (m *Mutex)unlockSlow(new int32) {
    if (new+mutexLocked)&mutexLocked == 0 {
        fatal("sync: unlock of unlocked mutex")
    }
    if new&mutexStarving == 0 {
        ...
    } else {
    // Передаем владение мьютексом следующему в очереди ожидания
        runtime_Semrelease(&m.sema, true, 1)
    }
}
```

m.unlockSlow()

```
old := new
for {
    if old>>mutexWaiterShift == 0 || old&(mutexLocked|mutexWoken|
mutexStarving) != 0 {
        return
    }
    new = (old - 1<<mutexWaiterShift) | mutexWoken
    if atomic.CompareAndSwapInt32(&m.state, old, new) {
        runtime_Semrelease(&m.sema, false, 1)
        return
    }
    old := m.state
}
```

sync.RWMutex

sync.RWMutex

```go
type RWMutex struct {
    w Mutex
    writerSem uint32
    readerSem uint32
    readerCount int32
    readerWait int32
}
```

sync.RWMutex

```go
func (rw *RWMutex)RLock()
func (rw *RWMutex)TryRLock()bool
func (rw *RWMutex)RUnlock()
func (rw *RWMutex)rUnlockSlow(r int32)
func (rw *RWMutex)Lock()
func (rw *RWMutex)TryLock() bool
func (rw *RWMutex)Unlock()
```

rw.RLock()

```go
func (rw *RWMutex)RLock() {
    if atomic.AddInt32(&rw.readerCount, 1) < 0 {
        runtime_SemacquireMutex(&rw.readerSem, false, 0)
    }
}
```

rw.TryRLock()

```go
func (rw *RWMutex) TryRLock()bool {
    for {
        c := atomic.LoadInt32(&rw.readerCount)
        if c < 0 {
            return false
        }
        if atomic.CompareAndSwapInt32(&rw.readerCount, c, c+1) {
            return true
        }
    }
}
```

rw. RUnlock()

```go
func (rw *RWMutex) RUnlock() {
    if r := rw.readerCount.Add(-1); r < 0 {
        rw.rUnlockSlow(r)
    }
}
```

rw.rUnlockSlow()

```go
func (rw *RWMutex)rUnlockSlow(r int32) {
    if r+1 == 0 || r+1 == -rwmutexMaxReaders {
        fatal("sync: RUnlock of unlocked RWMutex")
    }
    if atomic.AddInt32(&rw.readerWait, -1) == 0 {
        runtime_Semrelease(&rw.writerSem, false, 1)
    }
}
```

rw.Lock()

```
func (rw *RWMutex)Lock(){
    rw.w.Lock()
    r := atomic.AddInt32(&rw.readerCount, -rwmuxtexMaxReaders) +
rwmutexMaxReaders
    if r != 0 && atomic.AddInt32(&rw.readerWait, r) != 0 {
        runtime_SemacquireMutex(&rw.writerSem, false, 0)
    }
}
```

rw.TryLock()

```go
func (rw *RWMutex)TryLock() bool {
    if !rw.w.TryLock() {
        return false
    }
    if !atomic.CompareAndSwapInt32(&rw.readerCount, 0, -rwmutexMaxReaders) {
        rw.w.Unlock()
        return false
    }
    return true
}
```

rw.Unlock()

```go
func (rw *RWMutex)Unlock() {
    r := atomic.AddInt32(&rw.readerCount, rwmutexMaxReaders)
    if r >= rwmutexMaxReaders {
        fatal("sync: Unlock of unlocked RWMutex")
    }
    for i:=0; i < int(r); i++ {
        runtime_Semrelease(&rw.readerSem, false, 0)
    }
    rw.w.Unlock()
}
```

Бенчмарки

Бенчмарки

```go
func locker(wg *sync.WaitGroup, l sync.Locker) {
    defer wg.Done()
    result := int64(0)
    for i := 0; i < 100; i++ {
        l.Lock()
        result += counter
        time.Sleep(time.Microsecond)
        l.Unlock()
    }
}

func initGoroutines(
    readerCount int,
    writerCount int,
    readerLocker sync.Locker,
    writerLocker sync.Locker,
) {
    var wg sync.WaitGroup
    wg.Add(readerCount + writerCount)
    for i := 0; i < writerCount; i++ {
        go locker(&wg, writerLocker)
    }
    for i := 0; i < readerCount; i++ {
        go locker(&wg, readerLocker)
    }
    wg.Wait()
}
```

Бенчмарки

```
BenchmarkMutex1000Read1Write-8                2          574763706 ns/op
BenchmarkRWMutex1000Read1Write-8        1000000000               0.03777 ns/op
BenchmarkMutex1000Read100Write-8              2          535144450 ns/op
BenchmarkRWMutex1000Read100Write-8      1000000000               0.1186 ns/op
BenchmarkMutex1000Read1000Write-8             1         1710810753 ns/op
BenchmarkRWMutex1000Read1000Write-8           1         1016611676 ns/op
BenchmarkMutex1000Read10000Write-8            1         12786959556 ns/op
BenchmarkRWMutex1000Read10000Write-8          1         11490108373 ns/op
```

Бенчмарки

```go
func initMutexCounter(goroutinesCounter int) {
    var wg sync.WaitGroup
    var mu sync.Mutex
    wg.Add(goroutinesCounter)
    for i := 0; i < goroutinesCounter; i++ {
        go func() {
            defer wg.Done()
            mu.Lock()
            defer mu.Unlock()
            counter += 1
        }()
    }
    wg.Wait()
}

func initAtomicCounter(goroutinesCounter int) {
    var wg sync.WaitGroup
    wg.Add(goroutinesCounter)
    for i := 0; i < goroutinesCounter; i++ {
        go func() {
            defer wg.Done()
            atomic.AddInt64(&counter, 1)
        }()
    }
    wg.Wait()
}
```

Бенчмарки

```
BenchmarkAtomicCounter-8          1000000000          0.2326 ns/op
BenchmarkMutexCounter-8           1000000000          0.2391 ns/op
```

Примеры хорошего и плохого использования

```go
type Object struct {
  mu sync.Mutex
  name string
  surname string
  counter int
}

obj := Object{mu: sync.Mutex{}}

for i := 0; i< 100; i++ {
  go func(){
    ...
    obj.mu.Lock()
    obj.name = new_name
    obj.surname = new_surname
    obj.counter++
    obj.mu.Unlock()
    ...
  }()

}
```

```go
var mu sync.Mutex
counter := 0
for i := 0; i< 100; i++ {
  go func(){
    ...
    mu.Lock()
    counter++
    mu.Unlock()
    ...
  }()

}
```

Полезные ссылки

Email:
ilya+jugruspeaker@suhov.site