Package atomic go1.15.2 Latest

Published: **Sep 9, 2020** | License: BSD-3-Clause | Standard library

Overview

Package atomic provides low-level atomic memory primitives useful for implementing synchronization algorithms.

These functions require great care to be used correctly. Except for special, low-level applications, synchronization is better done with channels or the facilities of the sync package. Share memory by communicating; don't communicate by sharing memory.

The swap operation, implemented by the SwapT functions, is the atomic equivalent of:

```
old = *addr
*addr = new
return old
```

The compare-and-swap operation, implemented by the CompareAndSwapT functions, is the atomic equivalent of:

```
if *addr == old {
    *addr = new
    return true
}
return false
```

The add operation, implemented by the AddT functions, is the atomic equivalent of:

```
*addr += delta
return *addr
```

The load and store operations, implemented by the LoadT and StoreT functions, are the atomic equivalents of "return *addr" and "*addr = val".

func AddInt32

```
func AddInt32(addr *int32, delta int32) (new int32)
```

AddInt32 atomically adds delta to *addr and returns the new value.

func AddInt64

```
func AddInt64(addr *int64, delta int64) (new int64)
```

AddInt64 atomically adds delta to *addr and returns the new value.

func AddUint32

```
func AddUint32(addr *uint32, delta uint32) (new uint32)
```

AddUint32 atomically adds delta to *addr and returns the new value. To subtract a signed positive constant value c from x, do AddUint32(&x, ^uint32(c-1)). In particular, to decrement x, do AddUint32(&x, ^uint32(0)).

func AddUint64

```
func AddUint64(addr *uint64, delta uint64) (new uint64)
```

AddUint64 atomically adds delta to *addr and returns the new value. To subtract a signed positive constant value c from x, do AddUint64(&x, ^uint64(c-1)). In particular, to decrement x, do AddUint64(&x, ^uint64(0)).

func AddUintptr

```
func AddUintptr(addr *uintptr, delta uintptr) (new uintptr)
```

AddUintptr atomically adds delta to *addr and returns the new value.

func CompareAndSwapInt32

```
func CompareAndSwapInt32(addr *int32, old, new int32) (swapped bool)
```

CompareAndSwapInt32 executes the compare-and-swap operation for an int32 value.

func CompareAndSwapInt64

```
func CompareAndSwapInt64(addr *int64, old, new int64) (swapped bool)
```

CompareAndSwapInt64 executes the compare-and-swap operation for an int64 value.

func CompareAndSwapPointer

```
func CompareAndSwapPointer(addr *unsafe.Pointer, old, new unsafe.Pointer) (swapped bo
```

CompareAndSwapPointer executes the compare-and-swap operation for a unsafe.Pointer value.

func CompareAndSwapUint32

```
func CompareAndSwapUint32(addr *uint32, old, new uint32) (swapped bool)
```

CompareAndSwapUint32 executes the compare-and-swap operation for a uint32 value.

func CompareAndSwapUint64

```
func CompareAndSwapUint64(addr *uint64, old, new uint64) (swapped bool)
```

CompareAndSwapUint64 executes the compare-and-swap operation for a uint64 value.

func CompareAndSwapUintptr

```
func CompareAndSwapUintptr(addr *uintptr, old, new uintptr) (swapped bool)
```

CompareAndSwapUintptr executes the compare-and-swap operation for a uintptr value.

func LoadInt32

```
func LoadInt32(addr *int32) (val int32)
```

LoadInt32 atomically loads *addr.

func LoadInt64

```
func LoadInt64(addr *int64) (val int64)
```

LoadInt64 atomically loads *addr.

func LoadPointer

```
func LoadPointer(addr *unsafe.Pointer) (val unsafe.Pointer)
```

LoadPointer atomically loads *addr.

func LoadUint32

```
func LoadUint32(addr *uint32) (val uint32)
```

LoadUint32 atomically loads *addr.

func LoadUint64

```
func LoadUint64(addr *uint64) (val uint64)
```

LoadUint64 atomically loads *addr.

func LoadUintptr

```
func LoadUintptr(addr *uintptr) (val uintptr)
```

LoadUintptr atomically loads *addr.

func StoreInt32

```
func StoreInt32(addr *int32, val int32)
```

StoreInt32 atomically stores val into *addr.

func StoreInt64

```
func StoreInt64(addr *int64, val int64)
```

StoreInt64 atomically stores val into *addr.

func StorePointer

```
func StorePointer(addr *unsafe.Pointer, val unsafe.Pointer)
```

StorePointer atomically stores val into *addr.

func StoreUint32

```
func StoreUint32(addr *uint32, val uint32)
```

StoreUint32 atomically stores val into *addr.

func StoreUint64

```
func StoreUint64(addr *uint64, val uint64)
```

StoreUint64 atomically stores val into *addr.

func StoreUintptr

```
func StoreUintptr(addr *uintptr, val uintptr)
```

StoreUintptr atomically stores val into *addr.

func SwapInt32

```
func SwapInt32(addr *int32, new int32) (old int32)
```

SwapInt32 atomically stores new into *addr and returns the previous *addr value.

func SwapInt64

```
func SwapInt64(addr *int64, new int64) (old int64)
```

SwapInt64 atomically stores new into *addr and returns the previous *addr value.

func SwapPointer

```
func SwapPointer(addr *unsafe.Pointer, new unsafe.Pointer) (old unsafe.Pointer)
```

SwapPointer atomically stores new into *addr and returns the previous *addr value.

func SwapUint32

```
func SwapUint32(addr *uint32, new uint32) (old uint32)
```

SwapUint32 atomically stores new into *addr and returns the previous *addr value.

func SwapUint64

```
func SwapUint64(addr *uint64, new uint64) (old uint64)
```

SwapUint64 atomically stores new into *addr and returns the previous *addr value.

func SwapUintptr

```
func SwapUintptr(addr *uintptr, new uintptr) (old uintptr)
```

SwapUintptr atomically stores new into *addr and returns the previous *addr value.

type Value

```
type Value struct {
    // contains filtered or unexported fields
}
```

A Value provides an atomic load and store of a consistently typed value. The zero value for a Value returns nil from Load. Once Store has been called, a Value must not be copied.

A Value must not be copied after first use.

func (*Value) Load

```
func (v *Value) Load() (x interface{})
```

Load returns the value set by the most recent Store. It returns nil if there has been no call to Store for this Value.

func (*Value) Store

```
func (v *Value) Store(x interface{})
```

Store sets the value of the Value to x. All calls to Store for a given Value must use values of the same concrete type. Store of an inconsistent type panics, as does Store(nil).

BUGs

- On x86-32, the 64-bit functions use instructions unavailable before the Pentium MMX.

  On non-Linux ARM, the 64-bit functions use instructions unavailable before the ARMv6k core.

  On ARM, x86-32, and 32-bit MIPS, it is the caller's responsibility to arrange for 64-bit alignment of 64-bit words accessed atomically. The first word in a variable or in an allocated struct, array, or slice can be relied upon to be 64-bit aligned.