**Tools**

**Race detection**

Race conditions are most insidious and elusive programming errors. It often long after the code deployed to mass production

Go 1.1 includes the race detection a new tool for finding the race conditions in go code.

It is currently support linux and windows

The race detector is based on the c/C++ thread sanitizer runtime library. Which will used to detect many errorsin googles internal codes.

when -race command-line will be set the compiler instruments all memory access and record, and see how the memory accessed.

Race enables binaries will use 10 times CPU and memory

Commands

go test -race mypkg       // Test the package

go run -race mysrc.go   //Compile and run the program

go build -race mycmd   //build the command

go install -race mypkg  //install the package

package main

import "fmt"

func main() {

    done := make(chan bool)

    m := make(map[string]string)

    m["name"] = "world"

    go func() {

        m["name"] = "data race"

        done <- true

    }()

    fmt.Println("Hello,", m["name"])

    <-done

}

go run -race racy.go

 func main() {

11      start := time.Now()

12      var t \*time.Timer

13      t = time.AfterFunc(randomDuration(), func() {

14          fmt.Println(time.Now().Sub(start))

15          t.Reset(randomDuration())

16      })

17      time.Sleep(5 \* time.Second)

18  }

19

20  func randomDuration() time.Duration {

21      return time.Duration(rand.Int63n(1e9))

22  }

23

panic: runtime error: invalid memory address or nil pointer dereference

[signal 0xb code=0x1 addr=0x8 pc=0x41e38a]

goroutine 4 [running]:

time.stopTimer(0x8, 0x12fe6b35d9472d96)

    src/pkg/runtime/ztime\_linux\_amd64.c:35 +0x25

time.(\*Timer).Reset(0x0, 0x4e5904f, 0x1)

    src/pkg/time/sleep.go:81 +0x42

main.func·001()

    race.go:14 +0xe3

created by time.goFunc

    src/pkg/time/sleep.go:122 +0x48

The race detector shows the problem: an unsynchronized read and write of the variable t from different goroutines.

To fix the race condition we change the code to read and write the variable t only from the main goroutine:

10  func main() {

11      start := time.Now()

12      reset := make(chan bool)

13      var t \*time.Timer

14      t = time.AfterFunc(randomDuration(), func() {

15          fmt.Println(time.Now().Sub(start))

16          reset <- true

17      })

18      for time.Since(start) < 5\*time.Second {

19          <-reset

20          t.Reset(randomDuration())

21      }

22  }

Here the main goroutine is wholly responsible for setting and resetting the Timer t and a new reset channel communicates the need to reset the timer in a thread-safe way.