DD1339 Introduktion till datalogi

Namn: ***Hampus Fristedt***

Uppgift: VT8

Grupp nummer: 1

Övningsledare: Peter Boström

Betyg: ..... **Datum: ....** **Rättad av: ....**

1 Buggar

../bug1.go

```go
package main

import "fmt"

// I want this program to print "Hello world!", but it doesn't work.
// You need to send the string to the channel in another thread.
func main() {
    ch := make(chan string)
    go func() {
        ch <- "Hello world!"
    }()
    fmt.Println(<-ch)
}
```

../bug2.go

```go
package main

import "fmt"

// This program should go to 11, but sometimes it only prints 1 to 10.
/* The bug was that sometimes the main method exited before the Print
   method got a chance to print. In this version the for loop that
   prints is in the main function, so the program won't exit before all
   values have been printed. */
func main() {
    ch := make(chan int)
    go func() {
        for i := 1; i <= 11; i++ {
            ch <- i
        }
        close(ch)
    }()
    for n := range ch { // reads from channel until it's closed
        fmt.Println(n)
    }
}
```

2 Producers and consumers

2.1 Vad händer om man byter plats på satserna wgp.Wait() och close(ch) i slutet av main-funktionen?

Om vi stänger kanalen innan vi är säkra på att producenterna är klara finns det en stark risk att producenterna försöker skicka data till en stäng kanal, vilket ger fel i runtime.

2.2 Vad händer om man flyttar close(ch) från main-funktionen och i stället stänger kanalen i slutet av funktionen Produce?

Då kommer kanalen stängas när den första gorutinen som kör Produce är klar, vilket förstör för alla andra producenter som försöker skicka till kanalen.

2.3 Vad händer om man tar bort satsen close(ch) helt och hållet?

Då kommer gorutinerna som kör Consume att köra till programet avslutas eftersom kanalen aldrig stängs och Consume loopar tills kanalen stängs.

2.4 Vad händer om man ökar antalet konsumenter från 2 till 4?

Då paralliserar vi konsumerandet ytterligare och programmet blir snabbare.

2.5 Kan man vara säker på att alla strängar blir utskrivna innan programmet stannar?

Nej, eftersom programet avslutas så fort producenterna är klara, kanalen är stängd, och tiden har printats. Det är ingenting som garanterar att alla konsumenter blir klara.

../many2many.go

```go
// Stefan Nilsson 2013-03-13

// This is a testbed to help you understand channels better.
package main

import (
	"fmt"
	"math/rand"
	"strconv"
	"sync"
	"time"
)

func main() {
	// Use different random numbers each time this program is executed.
	rand.Seed(time.Now().Unix())

	const strings = 32
	const producers = 4
	const consumers = 2

	before := time.Now()
	ch := make(chan string)
	wgp := new(sync.WaitGroup)
	wgc := new(sync.WaitGroup)
	wgp.Add(producers)
	wgc.Add(consumers)
	for i := 0; i < producers; i++ {
		go Produce("p"+strconv.Itoa(i), strings/producers, ch, wgp)
	}
	for i := 0; i < consumers; i++ {
		go Consume("c"+strconv.Itoa(i), ch, wgc)
	}

	wgp.Wait() // Wait for all producers to finish.
	close(ch)
	wgc.Wait() // Wait for all consumers to finish. Must close channel
	   first to avoid deadlock.
	fmt.Println("time:", time.Now().Sub(before))
}

```

```go
// Produce sends n different strings on the channel and notifies wg when done.
func Produce(id string, n int, ch chan<- string, wg *sync.WaitGroup) {
    for i := 0; i < n; i++ {
        RandomSleep(100) // Simulate time to produce data.
        ch <- id + ":" + strconv.Itoa(i)
    }
    wg.Done()
}

// Consume prints strings received from the channel until the channel is closed.
func Consume(id string, ch <-chan string, wg *sync.WaitGroup) {
    for s := range ch {
        fmt.Println(id, "received", s)
        RandomSleep(100) // Simulate time to consume data.
    }
    wg.Done()
}

// RandomSleep waits for x ms, where x is a random number, 0 <= x < n,
// and then returns.
func RandomSleep(n int) {
    time.Sleep(time.Duration(rand.Intn(n)) * time.Millisecond)
}
```

3 Oracle

../oracle.go

```go
// Stefan Nilsson 2013-03-13

// This program implements an ELIZA-like oracle (en.wikipedia.org/wiki/ELIZA).
package main

import (
    "bufio"
    "fmt"
    "math/rand"
    "os"
    "strings"
    "time"
)

const (
    star   = "Pythia"
    venue  = "Delphi"
    prompt = "> "
)

func main() {
    fmt.Printf("Welcome to %s, the oracle at %s.\n", star, venue)
    fmt.Println("Your questions will be answered in due time.")

    oracle := Oracle()
    reader := bufio.NewReader(os.Stdin)
    for {
        fmt.Print(prompt)
        line, _ := reader.ReadString('\n')
        line = strings.TrimSpace(line)
        if line == "" {
            continue
        }
        fmt.Printf("%s heard: %s\n", star, line)
        oracle <- line // The channel doesn't block.
    }
}

// Oracle returns a channel on which you can send your questions to the oracle.
// You may send as many questions as you like on this channel, it never blocks.
// The answers arrive on stdout, but only when the oracle so decides.
// The oracle also prints sporadic prophecies to stdout even without being asked.
func Oracle() chan<- string {
    questions := make(chan string)
    answers := make(chan string)
    // TODO: Answer questions.
    go func() {
        for question := range questions {
            go prophecy(question, answers)
        }
    }()

    // TODO: Make prophecies.
    go func() {
        for {
            prophecy("", answers)
        }
    }()
    // TODO: Print answers.
    go func() {
        for answer := range answers {
            fmt.Println("\n" + answer)
            fmt.Print(prompt)
        }
    }()
    return questions
}

// This is the oracle's secret algorithm.
// It waits for a while and then sends a message on the answer channel.
// TODO: make it better.
func prophecy(question string, answer chan<- string) {
    // Keep them waiting. Pythia, the original oracle at Delphi,
    // only gave prophecies on the seventh day of each month.
    time.Sleep(time.Duration(20+rand.Intn(10)) * time.Second)

    // Find the longest word.
    longestWord := ""
    words := strings.Fields(question) // Fields extracts the words into a slice.
    for _, w := range words {
        if len(w) > len(longestWord) {
            longestWord = w
        }
    }

    // Cook up some pointless nonsense.
    nonsense := []string{
        "The moon is dark.",
        "The sun is bright.",
    }
    answer <- longestWord + "... " + nonsense[rand.Intn(len(nonsense))]
}

func init() { // Functions called "init" are executed before the main function.
    // Use new pseudo random numbers every time.
    rand.Seed(time.Now().Unix())
}
```