Matching

Removing the go statement from Seek will turn the program into a synchronous one, where all steps can be predicted. Anna is going to send a message to Bob, Cody to Dave, and Eva to no one.

Changing wait group declarations will throw an error because we don't pass the wait group by reference to Seek method and so the wait group inside Seek isn't able to communicate with the one in main. Two messages will be sent/received and the last one will fail as all go routines are asleep.

Removing the buffer will thrown an error because unbuffered channels accept sends only if there is a corresponding receive. There doesn't seem to be a clear receiver of the sent information through the channel.

By default channels are unbuffered, meaning that they will only accept sends (chan ¡-) if there is a corresponding receive (¡- chan) ready to receive the sent value. Buffered channels accept a limited number of values without a corresponding receiver for those values.¹

Removing default will not cause any problems in the current set up as it is constructed in such a way that one will be left unmatched. However, decreasing the size of the array of people, for example by removing *Eva* will throw a an error (a deadlock) because there is no one in the match channel left and it is still open. Thus, we are waiting for something that never arrives, which leads to a deadlock. A solution would be to add close(match) right after wg.Wait().

Julia

The program uses all CPUs. The speed improved in the order of 20s. It seems like some might argue that we should have go routines for each pixel, but I think that we should have four that can be spread out on 4 processors.

```
// Stefan Nilsson 2013-02-27
   // This program creates pictures of Julia sets
       (en.wikipedia.org/wiki/Julia\_set).
  package main
4
5
  import (
6
       "\operatorname{fmt}"
       "image"
8
       "image/color"
9
       "image/png"
10
       "log"
11
       "math/cmplx"
12
       " os "
13
       "runtime"
       "strconv"
15
       "sync"
16
       "time"
17
18
19
  type ComplexFunc func (complex128) complex128
20
   var Funcs [] ComplexFunc = [] ComplexFunc {
22
       func(z complex128) complex128 { return <math>z*z - 0.61803398875
23
       func(z complex128) complex128 \{ return z*z + complex(0, 1) \},
24
```

¹https://gobyexample.com/channel-buffering, accessed 2015.04.14

```
func(z complex128) complex128 { return z*z + complex(-0.835, -0.2321)
25
       func(z complex128) complex128 { return z*z + complex(0.45, 0.1428) },
       func (z complex128) complex128 { return z*z*z + 0.400 },
27
       func(z complex128) complex128 { return cmplx.Exp(z*z*z) - 0.621 },
       func(z complex128) complex128 { return (z*z+z)/cmplx.Log(z) +
           complex (0.268, 0.060) },
       func(z complex128) complex128 { return cmplx.Sqrt(cmplx.Sinh(z*z)) +
30
           complex(0.065, 0.122)},
  }
31
32
  func main() {
33
       // use all the power!
34
       numcpu := runtime.NumCPU()
       runtime.GOMAXPROCS(numcpu)
36
37
       start := time.Now()
38
39
       ch := make(chan error, len(Funcs)) // new channel
40
41
       wg := new(sync.WaitGroup)
       wg.Add(len(Funcs)) // the number of functions = no. of pictures.
       \quad \mathbf{for} \ n, \ fn := range \ Funcs \ \{
44
           go CreatePng ("picture-"+strconv. Itoa (n)+".png", fn, 1024, wg, ch)
45
       }
46
       wg. Wait()
48
       close (ch)
49
       for k := range ch \{
51
            if k != nil {
52
                log.Fatal(k)
53
54
55
       fmt. Println (time. Since (start))
56
  }
57
   // CreatePng creates a PNG picture file with a Julia image of size n \times n.
59
  func\ CreatePng (\,filename\ string\ ,\ f\ ComplexFunc\ ,\ n\ \textbf{int}\ ,\ wg\ *sync\ .WaitGroup\ ,
60
      ch chan error) \{ // need pointer to wg.
       file, err := os. Create (filename)
61
       if err != nil {
62
           ch <- err
63
           return
       defer file. Close()
66
       ch <- png. Encode (file, Julia (f, n))
67
68
       wg. Done()
69
       return
70
  }
71
   // Julia returns an image of size n \times n of the Julia set for f.
73
  func Julia (f ComplexFunc, n int) image.Image {
74
       bounds := image.Rect(-n/2, -n/2, n/2, n/2)
75
       img := image.NewRGBA(bounds)
```

```
s := float64(n / 4)
77
78
        // code duplication... not good.
79
80
        for i := bounds.Min.X; i < bounds.Max.X; i \leftrightarrow \{
81
            wg := new(sync.WaitGroup)
            wg.Add(4)
83
            go func() {
84
                 for j := bounds.Min.Y; j < bounds.Max.Y/4; j ++ \{
85
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
87
                      r := uint8(0)
                     g := uint8(0)
                     b := uint8 (n \% 32 * 8)
                     img. Set(i, j, color.RGBA\{r, g, b, 255\})
91
92
                 }
93
94
                 wg. Done()
95
             }()
96
            go func() {
                 for j := bounds.Max.Y / 4; j < 2*(bounds.Max.Y/4); j \leftrightarrow \{
99
100
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
101
                     r := uint8(0)
102
                     g := uint8(0)
103
                     b := uint8 (n \% 32 * 8)
104
                     img.Set(i, j, color.RGBA\{r, g, b, 255\})
106
107
                 wg. Done()
108
            }()
109
110
            go func() {
111
                 for j := 2 * (bounds.Max.Y / 4); j < 3*(bounds.Max.Y/4); j++ {
112
113
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
114
                     r := uint8(0)
115
                     g := uint8(0)
116
                     b := uint8 (n \% 32 * 8)
                     img.Set(i, j, color.RGBA\{r, g, b, 255\})
118
119
                 }
                 wg.Done()
121
             }()
122
123
            go func() {
124
                 for j := 3*(bounds.Max.Y/4) + (bounds.Max.Y \% 4); j <
125
                     bounds.Max.Y; j++ {
126
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                      r := uint8(0)
128
                     g := uint8(0)
129
                     b := uint8 (n \% 32 * 8)
130
                     img.Set(i, j, color.RGBA\{r, g, b, 255\})
131
```

```
132
133
                  wg.Done()
134
              }()
135
136
              wg. Wait()
138
139
        return img
140
   }
142
   // Iterate sets z_0 = z, and repeatedly computes z_n = f(z_{n-1}), n = 1,
143
    // until |z_n| > 2 or n = max and returns this n.
144
   func\ Iterate (f\ ComplexFunc\,,\ z\ complex128\,,\ max\ \mathbf{int}\,)\ (n\ \mathbf{int})\ \{
         for; n < \max; n ++ 
146
              if real(z)*real(z)+imag(z)*imag(z) > 4 {
147
148
149
              z = f(z)
150
151
        return
152
153
```

Weather Station

```
// Stefan Nilsson 2013-02-27
   // This program creates pictures of Julia sets
       (en.wikipedia.org/wiki/Julia\_set).
   package main
4
   import (
6
        "fmt"
        "image"
        "image/color"
        "image/png"
10
        "log"
11
        "math/cmplx"
12
        ^{"} os ^{"}
13
        "runtime"
14
        "strconv"
15
        "sync"
16
        "time"
17
18
19
   type \ ComplexFunc \ func (complex 128) \ complex 128
21
   var Funcs [] ComplexFunc = [] ComplexFunc{
22
        func(z complex128) complex128 { return z*z - 0.61803398875 },
23
        \label{eq:complex128} func \left(z \ complex128 \right) \ complex128 \ \left\{ \ \mathbf{return} \ z*z \ + \ complex \left(0 \,, \ 1\right) \ \right\},
        func (z complex128) complex128 { return z*z + complex(-0.835, -0.2321)
25
            },
        func (z complex 128) complex 128 \{ \textbf{return} \ z*z + complex (0.45, \ 0.1428) \},
26
        func (z complex128) complex128 { return z*z*z + 0.400 },
27
```

```
func(z complex128) complex128 \{ return cmplx.Exp(z*z*z) - 0.621 \},
28
       func(z complex128) complex128 { return (z*z+z)/cmplx.Log(z) +}
29
           complex (0.268, 0.060) },
       func(z complex128) complex128 { return cmplx.Sqrt(cmplx.Sinh(z*z)) +
30
           complex (0.065, 0.122) },
  }
31
32
  func main() {
33
       // use all the power!
34
       numcpu := runtime.NumCPU()
       runtime.GOMAXPROCS(numcpu)
36
37
       start := time.Now()
       ch := make(chan error, len(Funcs)) // new channel
40
41
       wg := new(sync.WaitGroup)
42
       wg.Add(len(Funcs)) // the number of functions = no. of pictures.
43
       for n, fn := range Funcs {
44
           go CreatePng("picture-"+strconv.Itoa(n)+".png", fn, 1024, wg, ch)
45
       wg. Wait()
48
       close (ch)
49
50
       for k := range ch {
51
           if k != nil {
52
                log. Fatal(k)
53
55
       fmt. Println (time. Since (start))
56
  }
57
58
  // CreatePng creates a PNG picture file with a Julia image of size n x n.
59
  func CreatePng(filename string, f ComplexFunc, n int, wg *sync.WaitGroup,
60
      ch chan error) { // need pointer to wg.
       file, err := os. Create (filename)
61
       if err != nil {
62
           ch <- err
63
           return
64
       defer file. Close()
66
       ch <- png.Encode(file, Julia(f, n))
67
       wg.Done()
69
       return
70
  }
71
72
  // Julia returns an image of size n x n of the Julia set for f.
  func Julia (f ComplexFunc, n int) image. Image {
74
       bounds := image.Rect(-n/2, -n/2, n/2, n/2)
75
       img := image.NewRGBA(bounds)
       s := \operatorname{float} 64 \left( n \ / \ 4 \right)
77
78
       // code duplication ... not good.
79
```

```
for i := bounds.Min.X; i < bounds.Max.X; i++ {
81
            wg := new(sync.WaitGroup)
82
            wg.Add(4)
83
            go func() {
84
                 for j := bounds.Min.Y; j < bounds.Max.Y/4; j ++ \{
85
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
87
                     r := uint8(0)
88
                     g := uint8(0)
89
                     b := uint8 (n \% 32 * 8)
                     img.Set(i, j, color.RGBA\{r, g, b, 255\})
91
92
                 }
                 wg. Done()
95
             }()
96
97
            go func() {
98
                 for j := bounds.Max.Y / 4; j < 2*(bounds.Max.Y/4); j \leftrightarrow \{
99
100
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                     r := uint8(0)
102
                     g := uint8(0)
103
                     b := uint8 (n \% 32 * 8)
104
                     img.Set(i, j, color.RGBA\{r, g, b, 255\})
105
106
                 }
107
                 wg.Done()
108
             }()
110
            go func() {
111
                 for j := 2 * (bounds.Max.Y / 4); j < 3*(bounds.Max.Y/4); j++ {
112
113
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
114
                     r := uint8(0)
115
                     g := uint8(0)
116
                     b := uint8 (n \% 32 * 8)
117
                     img. Set(i, j, color.RGBA\{r, g, b, 255\})
118
119
                 }
120
                 wg.Done()
             }()
122
123
            go func() {
                 for j := 3*(bounds.Max.Y/4) + (bounds.Max.Y \% 4); j <
125
                     bounds.Max.Y; j ++ \{
126
                     n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
127
                     r := uint8(0)
128
                     g := uint8(0)
129
                     b := uint8 (n \% 32 * 8)
130
                     img.Set(i, j, color.RGBA\{r, g, b, 255\})
132
133
                 wg.Done()
134
            }()
```

```
136
               wg.Wait()
137
         }
138
139
         return img
140
141
142
    //\  \  \, Iterate\  \  \, sets\  \  z\_0\  =\  z\,,\  \  \, and\  \  \, repeatedly\  \  \, computes\  \  z\_n\  =\  f(z\_\{n-1\})\,,\  \  n\  \  1\,,
143
    // until |z_n| > 2 or n = max and returns this n.
144
    func Iterate(f ComplexFunc, z complex128, max int) (n int) {
         for ; n < \max; n +++ {
146
               if real(z)*real(z)+imag(z)*imag(z) > 4  {
147
                    {\bf break}
148
149
               z = f(z)
150
151
         return
152
153
```