

Fundamentos de PPD

Exercícios - concorrência em Go

Fonte: material encontrado na Internet

Fernando Luís Dotti – Escola Politécnica - PUCRS

Concorrência em Go (repetindo)

Utilizamos Go nesta disciplina para exercitar concorrência

Optamos por Go por oferecer a abstração de canais (de forma condizente com nosso estudo de CSP)

Utilizaremos somente as abstrações de canais, semáforos e monitores. Demais mecanismos de sincronização oferecidos pela linguagem são secundários.

Exceto canais o uso de outros mecanismos de sincronização são comuns a outras linguagens

Concorrência em Go: **Processos (go rotinas)**

Go routine: um processo sequencial concorrente com os demais, uma thread

Comando **go <nomeDeFunção>** cria um novo processo executando o código da função

- Escopo local da função é individual para cada instância de gorotina executando-a
- Variáveis globais ou passadas por endereço são compartilhadas por go rotinas concorrentes (memória compartilhada)

Equivale a **criar thread** em qualquer linguagem multithreaded

Exercício:

Procure entender, executar e avaliar os resultados dos programas a seguir.

Concorrência em Go

Go rotinas

- **main** é um processo sequencial
- **go** cria outro processo sequencial executando a função especificada

Saída esperada ?

...algo como:

world

hello

hello

world

world

hello

hello

world

world

hello

```
package main
```

```
import (  
    "fmt"  
    "time"  
)
```

```
func say(s string) {  
    for i := 0; i < 5; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(s)  
    }  
}
```

```
func main() {  
    go say("world")  
    say("hello")  
}
```

Concorrência em Go: **Processos (go rotinas)**

Se o processo main() acabar,
todas as go rotinas criadas serão
terminadas antes de acabarem a
execução de seu código

Cabe ao programador garantir a
espera

Isto é comum a todas linguagens
multithreaded

Concorrência em Go

Go rotinas – terminação

- No exemplo anterior retire `time.Sleep`
- Execute várias vezes e avalie o resultado

Saída 1
??:

hello
hello
hello
hello
hello

Saída 2
??:

hello
hello
hello
hello
hello
world

Saída 3
??:

hello
world
world
world
world
world
hello
hello
hello
hello

```
package main

import (
    "fmt"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        fmt.Println(s)
    }
}

func main() {
    go say("world")
    say("hello")
}
```

Concorrência em Go

Go rotinas – terminação – solução análoga

- Esperando com waitGroups
- Isto é da biblioteca de Go e não tem análogo em outras linguagens
- Não precisa utilizar. Aqui para conhecimento.

Saída:


hello
world
world
world
world
hello
hello
hello
hello

```
package main

import (
    "fmt"
    "sync"
)

func say(s string, wg *sync.WaitGroup) {
    for i := 0; i < 5; i++ {
        fmt.Println(s)
    }
    wg.Done() // informa fim
}

func main() {
    var waitgroup sync.WaitGroup
    waitgroup.Add(2) // quantos esperar
    go say("world", &waitgroup)
    go say("hello", &waitgroup)
    // aqui temos 3 processos concorrentes:
    // duas instancias de say, e o main
    waitgroup.Wait() // espera fim de todos
}
```



Concorrência em Go

Go rotinas – velocidades relativas

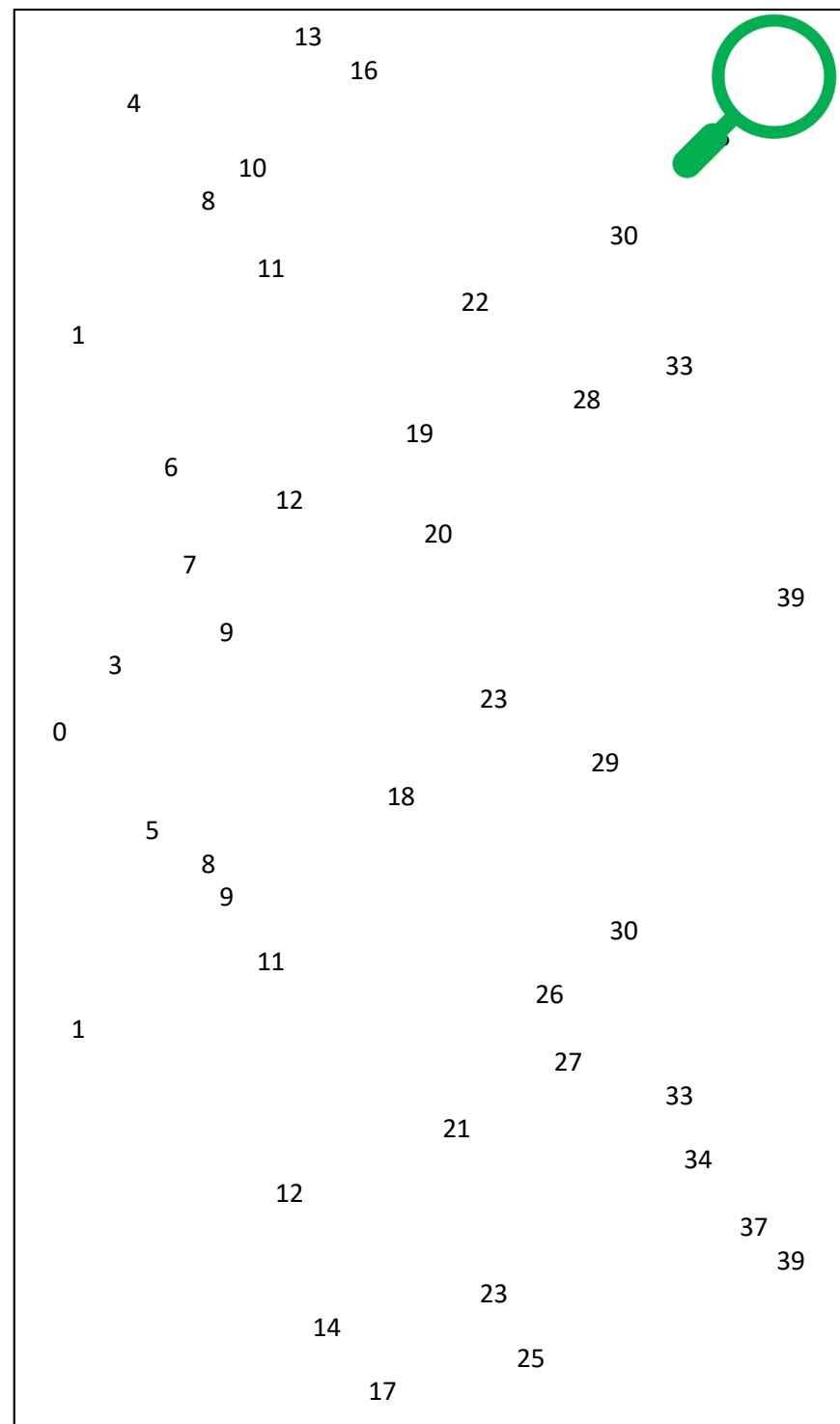
- veja extrato de saída do programa, ao lado
- a cada print, cada processo volta a sleep

```
package main
import (
    "fmt"
    "time"
)
var N int = 40

func geraNespacos(n int) string{ // somente auxiliar
    s := " " // gera String com i espacos
    for j := 0; j < n; j++ { s = s + " " }
    return s
}

func funcaoA(id int, s string) { // loop de cada processo id
    for {
        fmt.Println(s, id)
    }
}

func main() {
    for i := 0; i < N; i++ { // criação de processos
        go funcaoA(i, geraNespacos(i))
    }
    for true {
        time.Sleep(100 * time.Millisecond)
    }
}
```



Concorrência em Go

Go rotinas – velocidades relativas

- veja extrato de saída ao lado
- o mesmo processo executa várias iterações do seu loop

Não podemos fazer suposições sobre a velocidade relativa dos processos

Depende do escalonador

```
package main
import (
    "fmt"
    "time"
)
var N int = 40

func geraNespacos(n int) string{ // somente auxiliar
    s := " " // gera String com i espacos
    for j := 0; j < n; j++ { s = s + " " }
    return s
}

func funcaoA(id int, s string) { // loop de cada processo
    for {
        fmt.Println(s, id)
    }
}

func main() {
    for i := 0; i < N; i++ { // criação de processos
        go funcaoA(i, geraNespacos(i))
    }
    for true {
        time.Sleep(100 * time.Millisecond)
    }
}
```

