Laboratório de Programação Concorrente

Lab10 - Go Select - 25.1

Objetivo

O objetivo deste laboratório é que vocês explorem a construção de programas concorrentes em Go que precisam aguardar múltiplos canais simultaneamente, utilizando a instrução *select*. Com isso, esperamos que vocês observem ganhos de desempenho e pratiquem o design de sistemas mais responsivos.

Descrição

Considere que você está implementando um sistema de monitoramento de serviços distribuídos. Cada serviço responde com um tempo variável, e seu programa deve coletar respostas concorrentes para então decidir o que fazer. O programa terá de lidar com múltiplos produtores (simulando serviços) e um consumidor central (o monitor).

Etapas Incrementais

Etapa 1 – Produtor Simples

- Implemente a função exec(maxSleepMs int) int, que:
  - Dorme por um tempo aleatório entre 0 e maxSleepMs milissegundos.
  - Retorna esse valor de tempo dormido.
- Teste chamando essa função diretamente no main.

Etapa 2 – Produtor Concorrente

- Implemente uma função startProducer(maxSleepMs int) <-chan int que:
  - Cria uma goroutine que executa exec(maxSleepMs) 200 vezes.
  - Envia os resultados em um canal.
  - Retorna esse canal imediatamente.
- No main, crie dois produtores e receba valores de cada um sequencialmente. Considere como maxSleepMs 250 e 700, respectivamente, para o prod1 e prod2.
  - Pergunta para reflexão: o tempo de execução depende da ordem de leitura dos canais?

Etapa 3 – Uso de select

- Modifique o main para usar select, permitindo que os valores sejam consumidos assim que qualquer produtor tiver algo disponível.
- Compare o tempo total de execução com a versão da Etapa 2.

Etapa 4 – Timeout

- Acrescente ao main um mecanismo de timeout:
    - Se nenhum produtor responder em até 500ms, imprima "timeout" e continue aguardando.
    - Use time.After junto com select.

Para gerar números aleatórios, por exemplo entre 0 e maxSleepMs, use a biblioteca abaixo:

```
"math/rand"

rand.Seed(42)
for {
     v := rand.Intn(maxSleepMs)
}
```

Visão geral do código base

Aqui temos o pdf deste laboratório.

https://github.com/giovannifs/fpc/tree/master/2025.1/Lab10

Todas as soluções devem ser desenvolvidas no diretório src dentro do Lab10, um arquivo para cada etapa do lab. Seguem os nomes:

- lab10-e1.go
- lab10-e2.go
- lab10-e3.go
- lab10-e4.go

Prazo

16/set/25 16:00

Entrega

Você deve criar e manter um repositório privado no GitHub com a sua solução. No entanto, a entrega do laboratório deverá ser realizada por meio de submissão online utilizando o script submit-answer.sh, disponibilizado na estrutura de arquivos do próprio laboratório. Uma vez que você tenha concluído sua resposta, seguem as instruções:

1) Crie um arquivo lab10_matr1_matr2.tar.gz somente com o “src” do repositório que vocês trabalharam. Para isso, supondo que o diretório raiz de seu repositório privado chama-se lab8_pc, você deve executar:
   tar -cvzf lab10_matr1_matr2.tar.gz lab10_pc/src

2) Submeta o arquivo lab10_matr1_matr2.tar.gz usando o script submit-answer.sh, disponibilizado no mesmo repositório do laboratório:
   bash submit-answer.sh lab10 path/lab10_matr1_matr2.tar.gz

Lembre-se que você deve manter o seu repositório privado no GitHub para fins de comprovação em caso de problema no empacotamento ou transmissão online. Alterações no código realizadas após o prazo de entrega não serão analisadas.

Select

Considere, em um sistema grande, que módulos desse sistema se integram através de canais. Por exemplo, um gateway que se conecta à dois serviços HTTP (além de tratar requisições de seus próprios clientes).

Nesse caso, digamos que há 3 canais. O gateway deve reagir, quando receber quando evento em um desses canais. Como ficaria o "main loop" do gateway?

Forma Geral do Select

```
select {
    case <- ch1:
        // executa o receive em ch1 e descarta o valor
    case x := <-ch2:
        // executa o receive em ch2 e atribui em x
    case ch3 <- y:
        // executa send em ch3
    default:
        // claúsula opcional
}
```

- Cada case descreve uma comunicação possível e um bloco de declarações associado com cada uma das comunicações especificadas.
- O select esperará que uma das comunicações esteja pronta para ocorrer e em seguida, executa o bloco de declarações associado. As demais comunicações não acontecerão.
- Caso múltiplas comunicações estiverem prontas para ocorrer, uma delas é escolhida aleatoriamente

Timeout o Select

```
select {
      case v <-ch1:
            // executa receive em ch1 e atribui em v
      case v <-ch2:
            // executa receive em ch2 e atribui em v
      case <-time.After(500 * time.Millisecond):
            // trecho executado caso não execute o receive em nenhum
            // dos canais por 500ms
}
```