TPD: Algoritmos Distribuídos de Eleição

Grupo: José Junior Borges Monteiro e Loannis Lucas Nicou Canteiro

Na computação distribuída, muitas vezes é necessária a escolha de um processo específico para coordenar tarefas ou desempenhar alguma função em particular. Os algoritmos usados para escolher um coordenador são conhecidos como algoritmos de eleição de líder. Nesses algoritmos, muitas vezes não importa qual o processo é escolhido como líder, desde que alguém seja eleito. Contudo, todos os processos participantes devem concordar com a escolha do novo líder. Um exemplo prático em que é utilizado eleição de líder é em servidores que usam replicação passiva(Harp).

Para este trabalho foi utilizado o algoritmo em anel para realizar as eleições, este algoritmo realiza a eleição considerando uma rede de sobreposição em anel, o que significa que os processos se comunicam ao redor de um anel no sentido horário. Para esse algoritmo, é considerado que cada processo tenha uma espécie de "força", ou seja, um identificador (ID), onde o processo que possui a maior "força" (identificador) dentre todos os processos que não estejam em estado de falha pode ser eleito o líder.

Neste trabalho foram utilizados 3 processos, cada um contendo o seu id e os canais de comunicação com o processo à esquerda (entrada) e a direita (saída), para realizar a comunicação foi utilizado uma struct mensagem contendo o tipo da mensagem, podendo ser eleição, votação, confirmação e finalização e o corpo responsável por indicar estados de cada processo pelo seu ID. Para simular as falhas que ocorreriam normalmente em uma aplicação real foi utilizado uma função de controle, responsável por bloquear e recuperar um processo, convocar novas eleições e confirmar o novo líder.

Para simplificar a compreensão dos códigos foi utilizado os códigos do HTTP, sendo 500 para processo indisponível, 400 para falhas, 200 para sucesso e 100 para sinalizar um valor default.

Para simular uma falha foi criado o método que busca o processo líder e atribui ao corpo o código 500.

Quando o controle inicia o processo de eleição ele altera o tipo para 1 e ele indica o processo que irá iniciar escrevendo no seu canal de entrada e assim liberando o processo para executar a função, este processo inicial cairá no primeiro caso do switch, irá alterar o tipo da mensagem para 2 indicando que está em processo de votação posteriormente irá votar e repassar a mensagem para o próximo processo do anel.

Os demais processos cairão no caso 2 onde irão realizar a votação e quando for finalizado o processo que iniciou a votação irá utilizar o algoritmo para determinar o novo líder. Para este trabalho foi utilizado como algoritmo de escolha do líder o processo com maior ID.

Após definido o novo líder é passado uma mensagem para todos os processos do anel a fim de notificar e confirmar o novo líder.

Ademais, foi implementado também a função recuperaProcesso responsável por recuperar os processos inativos, para isso é necessário que o controle passe o id do processo que se deseja recuperar e a mensagem e o algoritmo irá verificar se o processo está de fato inativo e irá alterar o seu corpo para o valor default, indicando que ele está ativo novamente.

Primeira eleição com o processo 2 inativo

```
   Anel de processos criado
   Processo controlador criado

Controle: eleicao enviada

 0: recebi mensagem do tipo 1, com a solicitação de uma eleição para um novo coordenador 0: enviei mensagem do tipo 2, para iniciar a votação
 1: recebi mensagem do tipo 2, com o corpo [ 0, 100, 500 ]
 1: Realizei a votação e enviei para o próximo anel
 0: enviei mensagem do tipo 3, iniciando a confirmação do novo coordenador de ID = 1
 1: recebi mensagem do tipo 3, informando o novo coordenador de ID = 1
 0: Eleição confirmada, enviando confirmação para o controlador
Controle: confirmação 200
```

Segunda Eleição com o processo 1 e 2 inativo

```
Controle: eleicao enviada

 0: recebi mensagem do tipo 1, com a solicitação de uma eleição para um novo coordenador 0: enviei mensagem do tipo 2, para iniciar a votação
 0: enviei mensagem do tipo 3, iniciando a confirmação do novo coordenador de ID = 0
 0: Eleição confirmada, enviando confirmação para o controlador
Controle: confirmação 200
```

Processo de ID 2 recuperado e Terceira eleição com o processo 1 inativo

```
Controle: Processo recuperado com sucesso!
Controle: eleicao enviada

 2: recebi mensagem do tipo 1, com a solicitação de uma eleição para um novo coordenador 2: enviei mensagem do tipo 2, para iniciar a votação
 0: recebi mensagem do tipo 2, com o corpo [ 100, 500, 2 ]
 0: Realizei a votação e enviei para o próximo anel
 2: enviei mensagem do tipo 3, iniciando a confirmação do novo coordenador de ID = 2
 0: recebi mensagem do tipo 3, informando o novo coordenador de ID = 2
 2: Eleição confirmada, enviando confirmação para o controlador
Controle: confirmação 200

Controle: Processo finalizado!!
```

```
package main

import (
  "fmt"
  "math"
  "sync"
)

type mensagem struct {
  tipo  int
  corpo [3]int
}

var (
  mutex           sync.Mutex
  simulation_time int        = 0
  chans                      = []chan mensagem{
     make(chan mensagem),
     make(chan mensagem),
     make(chan mensagem),
  }
  pacote_eleicao mensagem
  controle       = make(chan int)
  wg             sync.WaitGroup
)

func ElectionControler(in chan int) {
  defer wg.Done()
  var temp mensagem
  inicializaMensagem(&temp)

  // Primeira Falha = Task com ID 2 falha
  simulaFalha(&temp)
  temp.tipo = 1
  fmt.Printf("Controle: eleicao enviada \n\n")
  chans[2] <- temp                                  // pedir eleição
  fmt.Printf("Controle: confirmação %d\n\n", <-in) // receber e imprimir
confirmação

  // Processo 1 é o novo coordenador
  // Segunda Falha = Task com ID 1 falha

  restauraValores(&temp)
  simulaFalha(&temp)
  temp.tipo = 1
```

```
  fmt.Printf("Controle: eleicao enviada \n\n")
  chans[2] <- temp                                  // pedir eleição
  fmt.Printf("Controle: confirmação %d\n\n", <-in) // receber e imprimir
confirmação

  recuperaProcesso(2, &temp)
  restauraValores(&temp)
  temp.tipo = 1
  fmt.Printf("Controle: eleicao enviada \n\n")
  chans[0] <- temp                                  // pedir eleição para
retornar o antigo coordenador
  fmt.Printf("Controle: confirmação %d\n\n", <-in) // receber e imprimir
confirmação

  restauraValores(&temp)
  temp.tipo = 4
  chans[2] <- temp

  fmt.Printf("Controle: Processo finalizado!!")
}

func restauraValores(m *mensagem) {
  for i, _ := range m.corpo {
     if m.corpo[i] != 500 {
        m.corpo[i] = 100
     }
  }
}

func recuperaProcesso(id int, m *mensagem) {
  if m.corpo[id] == 500 {
     fmt.Println("Controle: Processo recuperado com sucesso!")
     m.corpo[id] = 100
  } else {
     fmt.Println("Controle: Falha ao realizar recuperação, processo já
está ativo!")
  }
}

func simulaFalha(m *mensagem) {
  var indiceMaior = buscaMaiorValor(m)
  m.corpo[indiceMaior] = 500 // 500 = falha
}

func buscaMaiorValor(m *mensagem) int {
  var maior = math.MinInt8
```

```
  var indiceMaior = 400
  for i, valor := range m.corpo {
     if valor != 500 && valor >= maior && valor <= 100 {
        maior = valor
        indiceMaior = i
     }
  }
  return indiceMaior
}

func ehUltimo(m *mensagem, id int) bool {
  var current = 200
  for i, valor := range m.corpo {
     if valor != 200 && i != id {
        current = -1
     }
  }
  return current == 200
}

func inicializaMensagem(m *mensagem) {
  m.corpo[0] = 100
  m.corpo[1] = 100
  m.corpo[2] = 100
}

func ElectionStage(TaskId int, in chan mensagem, out chan mensagem) {
  for {
     temp := <-in // le do canal
     if temp.corpo[TaskId] == 500 {
        if temp.tipo == 4 {
           out <- temp
           wg.Done()
           break
        }
        out <- temp
     } else {
        switch temp.tipo {
        case 1:
           {
              fmt.Printf("%2d: recebi mensagem do tipo %d, com a
solicitação de uma eleição para um novo coordenador", TaskId, temp.tipo)

              temp.tipo = 2 // mensagem do tipo votação
              fmt.Printf("%2d: enviei mensagem do tipo %d, para iniciar
a votação\n", TaskId, temp.tipo)
```

```
            temp.corpo[TaskId] = TaskId
            out <- temp
            // aguarda passar por todos os sistemas do anel
            temp = <-in

            // vencedor da eleição e passado pela mensagem na primeira
posição do array
            var coordenador = buscaMaiorValor(&temp)
            temp.corpo[0] = coordenador
            temp.tipo = 3
            fmt.Printf("%2d: enviei mensagem do tipo %d, iniciando a
confirmação do novo coordenador de ID = %d\n", TaskId, temp.tipo,
coordenador)

            out <- temp
            // aguarda passar por todos os sistemas do anel
            temp = <-in

            // Envia confirmação da eleição para o controlador
            fmt.Printf("%2d: Eleição confirmada, enviando confirmação
para o controlador\n", TaskId)
            controle <- 200
         }
      case 2:
         {
            fmt.Printf("%2d: recebi mensagem do tipo %d, com o corpo [
%d, %d, %d ]\n", TaskId, temp.tipo, temp.corpo[0], temp.corpo[1],
temp.corpo[2])

            temp.corpo[TaskId] = TaskId
            fmt.Printf("%2d: Realizei a votação e enviei para o
próximo anel \n", TaskId)
            out <- temp
         }
      case 3:
         {
            var coordenador = temp.corpo[0]
            fmt.Printf("%2d: recebi mensagem do tipo %d, informando o
novo coordenador de ID = %d\n", TaskId, temp.tipo, coordenador)
            out <- temp
         }
      case 4:
         {
            if ehUltimo(&temp, TaskId) {
               wg.Done()
```

```
                break
            }
            temp.corpo[TaskId] = 200
            out <- temp
            wg.Done()
            break
         }
      default:
         {
            fmt.Printf("%2d: não conheço este tipo de mensagem\n",
TaskId)
         }
      }
   }
}

func main() {

  wg.Add(4) // Add a count of four, one for each goroutine

  // criar os processo do anel de eleicao

  go ElectionStage(0, chans[2], chans[0])
  go ElectionStage(1, chans[0], chans[1])
  go ElectionStage(2, chans[1], chans[2])

  fmt.Println("\n   Anel de processos criado")

  // criar o processo controlador

  go ElectionControler(controle)

  fmt.Println("   Processo controlador criado\n")

  wg.Wait() // Wait for the goroutines to finish\
}
```