Alexandre Júnior Lelis Rodrigues
Matrícula: -
Código fonte comentado:

```
package main
/*
Os tempos dos produtores, consumidores e da limpeza do buffer estão fora de sincronia
para mostrar que isso não faz diferença

não entendi claramente se eu devo enviar a frase inteira palavra por palavra de uma vez e
liberar o semáforo
ou enviar uma palavra e liberar o semáforo, porém trocar essa lógica é bem simples
*/
import (
	"crypto/rand"
	"fmt"
	"math/big"
	"sync"
	"time"
)

const (
	MAX      = 4 // A quantidade de produtores e consumidores (4 para cada nesse caso)
	wordSize = 4 // tamanho das palavras que serão geradas
)

var (
	mut          sync.Mutex        // semáforo
	finished     bool      = false // verdadeira se o programa não tiver mais nada a fazer
	producerDead [MAX]bool         // verdadeiro para cada produtor que terminou sua
produção
)

/*
Decidi implementar uma lista encadeada para melhorar minhas habilidades em GO, cada
nó contém a palavra a ser lida (word), quantos ainda faltam ler (toRead), se um produtor X
leu
esse nó (eachRead) e o ponteiro para o próximo nó (next)
*/
type Node struct {
	word     string
	toRead   int
	eachRead [MAX]bool
	next     *Node
}
type linkedList struct {
	head *Node
```

```
}

// declaro a lista globalmente
var list linkedList

// Insere sempre no final da lista encadeada
func (ll *linkedList) Insert(word string) {

	newNode := &Node{word: word, toRead: MAX, next: nil}
	/*toRead vem com o máximo de threads
	  e irá ser decrementado quando algum consumidor o lê*/
	if ll.head == nil {
		ll.head = newNode
	} else {
		current := ll.head
		for current.next != nil {
			current = current.next
		}
		current.next = newNode
	}
}

// Deleta um nó da lista com base na palavra, traversa ela até encontrar
func (ll *linkedList) Delete(word string) {
	if ll.head == nil {
		return
	}
	if ll.head.word == word {
		ll.head = ll.head.next
		return
	}
	prev := ll.head
	for prev.next != nil {
		if prev.next.word == word {
			prev.next = prev.next.next
			return
		}
		prev = prev.next
		//go possui coleta de lixo, não é preciso dar free no nó excluido
	}
}

// Função gera strings aleartorias (Peguei de um outro projeto meu)
func randomString(length int) string {
	const charset =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

	randomString := make([]byte, length)
```

```
	charsetLength := big.NewInt(int64(len(charset)))

	for i := 0; i < length; i++ {
		randomIndex, err := rand.Int(rand.Reader, charsetLength)
		if err != nil {
			panic(err)
		}
		randomString[i] = charset[randomIndex.Int64()]
	}

	return string(randomString)
}

func producer(id int) {
	var frase [4]string
	//gerando uma frase com 4 palavras geradas aleatoriamente

	for i := 0; i < 4; i++ {
		frase[i] = randomString(wordSize)
	}
	// para fins de debug eu dou print na frase inteira, porém isso não consta nos
requisitos
	fmt.Println("frase: " + frase[0] + " " + frase[1] + " " + frase[2] + " " + frase[3])
  // Versão 1: Envia a frase inteira palavra por palavra e depois libera o semáforo
  // Dessa forma a frase fica na sequência correta, na versão 2 não, muito mais lenta que a
versão 2
	mut.Lock()
  for _, word := range frase {
    list.Insert(word)
    time.Sleep(time.Millisecond * 500) // espera meio segundo
  }
  mut.Unlock()
  /*versão 2: Enviando uma palavra por vez e abrindo o semáforo
  for _, word := range frase {
		//envia uma palvra por vez ao buffer e espera 1 segundo, semáforo cerca a
inserção na lista
		mut.Lock()
		list.Insert(word)
		mut.Unlock()
		time.Sleep(time.Second) //espera 1 segundo
	}
*/
	// se chegou aqui, é porque terminou de enviar sua frase inteira, logo, este produtor
	// terminou sua função e morreu
	producerDead[id] = true
}

/*
```

```
separei a função de consumir em si para poder utilizar recursão.
O código checa se o nó já foi lido por esse consumidor X, caso não tenha sido lido
lê, caso contrário checa o próximo, se o nó for nulo retorna no começo
*/
func consume(node *Node, id int) {
	if node == nil {
		return
	}

	if node.eachRead[id] {
		consume(node.next, id)
	} else {
		node.eachRead[id] = true	// Esse nó já leu, logo vira verdadeiro
		node.toRead--			// Decrementa quantos ainda faltam ler
		fmt.Printf(node.word+" p%d ", id) //printa a palavra mais o ID
		if node.toRead == 0 {
			fmt.Println() // Quebra a linha quando todos os consumidores leram a
palavra X
		}
	}
}

/*
  waitGroup nesse caso é utilizado para fazer a thread principal esperar os
  consumidores terminarem para poder finalizar o código

  enquanto a thread main não sinalizar que acabou, continua.const
  Se a lista encadeada não estiver vazia no momento, chama a função acima de consumir
  espera 200 milisegundos
*/
func consumer(id int, wg *sync.WaitGroup) {
	defer wg.Done()
	for !finished {
		if list.head != nil {
		  //semáforo ativa na hora de consumir
			mut.Lock()
			consume(list.head, id)
			mut.Unlock()
			time.Sleep(time.Millisecond * 200)
		}
	}
}
/*
Em go, uma função é concorrente quando se coloca o prefixo go antes de chamar ela,
no caso estou criando 4 produtores e 4 consumidores, esse numero pode ser facilmente
modificado ao alterar a variável global MAX
*/
func main() {
```

```
	var consumerGroup sync.WaitGroup // Declaro a lista de espera dos consumidores

	consumerGroup.Add(MAX)

	for i := 0; i < MAX; i++ {
		go producer(i)
	}

	for i := 0; i < MAX; i++ {
		go consumer(i, &consumerGroup)
	}


	for !finished {
	  // Acaba quando todos os produtores estiverem mortos e a lista estiver vazia
		if !producing() && list.head == nil {
			finished = true
			break
		} else {
		  //Lista de palavras que já foram lidas recebidas da função abaixo
			words_to_clean := findFinished()
			//se a lista de palavras não for nula, ativa o semáforo e retira da lista
			//cada um que já foi lido
			if len(words_to_clean) != 0 {
				mut.Lock()
				for _, word := range words_to_clean {
					list.Delete(word)

				}
				mut.Unlock()
			}
			time.Sleep(time.Second * 2) // espera 2 segundos
		}
	}
	//espera os consumidores terminarem para dar fim ao código
	consumerGroup.Wait()
}

// traversa a lista encadeada e procura todos os nós que não faltam consumidor a ler
(toRead = 0)
// retorna uma lista com todas as palavras que já foram lidas por todos
func findFinished() (words []string) {
	node := list.head
	for node != nil {
		if node.toRead == 0 {
			words = append(words, node.word)
		}
```

```
                node = node.next
        }
        return words
}

// Se existir algum produtor que ainda está vivo, retorna verdadeiro, caso contrário falso
func producing() bool {
        for i := 0; i < MAX; i++ {
                if !producerDead[i] {
                        return true
                }
        }
        return false
}
```

Habilitei o print das frases pelo produtor apenas para mostrar que as mensagens estão enviadas corretamente.

```
PS C:\Users\apns\code\College\sistemas operacionais\ex 3 - sinc> go run .
frase: nB11 DeZY ftoS Np6c
frase: qnUK tfLl HJJI lLcB
frase: bP2C NIK3 6R4j V5xF
frase: baAm Dq2D nReu 8n93
nB11 p1|nB11 p2|nB11 p3|nB11 p0|
DeZY p0|DeZY p3|DeZY p1|DeZY p2|
ftoS p1|ftoS p3|ftoS p0|ftoS p2|
Np6c p2|Np6c p3|Np6c p1|Np6c p0|
qnUK p0|qnUK p3|qnUK p2|qnUK p1|
tfLl p1|tfLl p3|tfLl p0|tfLl p2|
HJJI p2|HJJI p0|HJJI p1|HJJI p3|
lLcB p3|lLcB p0|lLcB p2|lLcB p1|
bP2C p0|bP2C p1|bP2C p3|bP2C p2|
NIK3 p2|NIK3 p1|NIK3 p3|NIK3 p0|
6R4j p0|6R4j p1|6R4j p2|6R4j p3|
V5xF p3|V5xF p1|V5xF p0|V5xF p2|
baAm p2|baAm p1|baAm p3|baAm p0|
Dq2D p0|Dq2D p1|Dq2D p2|Dq2D p3|
nReu p3|nReu p1|nReu p0|nReu p2|
8n93 p1|8n93 p2|8n93 p3|8n93 p0|
```

2) para mostrar que a quantidade não faz diferença, abaixo foi feito com 7 produtores e 7 consumidores, diminui o tamanho das palavras para não ficar muito grande

```
PS C:\Users\apns\code\College\sistemas operacionais\ex 3 - sinc> go run .
frase: 4v RG rv tS
frase: yC Rr KF uW
frase: vO HI dd Tu
frase: GH 5f 9I 3f
```

frase: NK Bz xl Cz
frase: e0 bC Ju QB
frase: 3e Oi hi qG
4v p6|4v p2|4v p0|4v p3|4v p4|4v p1|4v p5|
RG p5|RG p6|RG p2|RG p0|RG p3|RG p4|RG p1|
rv p1|rv p2|rv p5|rv p6|rv p0|rv p4|rv p3|
tS p3|tS p4|tS p1|tS p0|tS p5|tS p2|tS p6|
yC p6|yC p1|yC p3|yC p4|yC p0|yC p5|yC p2|
Rr p2|Rr p5|Rr p1|Rr p0|Rr p3|Rr p4|Rr p6|
KF p1|KF p2|KF p5|KF p0|KF p3|KF p6|KF p4|
uW p3|uW p2|uW p5|uW p1|uW p0|uW p4|uW p6|
vO p0|vO p2|vO p3|vO p5|vO p1|vO p4|vO p6|
HI p5|HI p2|HI p0|HI p3|HI p1|HI p4|HI p6|
dd p1|dd p5|dd p0|dd p2|dd p3|dd p4|dd p6|
Tu p3|Tu p5|Tu p1|Tu p0|Tu p2|Tu p6|Tu p4|
GH p2|GH p0|GH p3|GH p5|GH p1|GH p6|GH p4|
5f p0|5f p1|5f p3|5f p5|5f p2|5f p6|5f p4|
9I p0|9I p2|9I p1|9I p3|9I p5|9I p4|9I p6|
3f p5|3f p1|3f p0|3f p2|3f p3|3f p6|3f p4|
NK p3|NK p1|NK p5|NK p0|NK p2|NK p4|NK p6|
Bz p2|Bz p5|Bz p0|Bz p1|Bz p3|Bz p6|Bz p4|
xl p3|xl p5|xl p2|xl p1|xl p0|xl p4|xl p6|
Cz p0|Cz p3|Cz p1|Cz p5|Cz p2|Cz p4|Cz p6|
e0 p5|e0 p3|e0 p0|e0 p2|e0 p1|e0 p4|e0 p6|
bC p0|bC p1|bC p2|bC p3|bC p5|bC p6|bC p4|
Ju p5|Ju p2|Ju p0|Ju p3|Ju p1|Ju p4|Ju p6|
QB p3|QB p2|QB p5|QB p0|QB p1|QB p6|QB p4|
3e p1|3e p3|3e p0|3e p5|3e p2|3e p6|3e p4|
Oi p2|Oi p0|Oi p5|Oi p3|Oi p1|Oi p4|Oi p6|
hi p0|hi p1|hi p3|hi p5|hi p2|hi p6|hi p4|
qG p1|qG p2|qG p5|qG p0|qG p3|qG p4|qG p6|