# 生产者消费者模型

## 1. 问题介绍

有界缓冲区有20个存储单元，分别存储1-20这20个整形数，生产者向其中存放整形数，消费者从中拿取整形数，所有存放和拿取的操作均在缓冲区末尾进行，生产者和消费者的数量均至少有两个，每次向缓冲区中存放或拿取整形数都只允许一个操作者进行，且当缓冲区满时，生产者均被堵塞，直到有消费者拿出资源，缓冲区有空时才会继续放入，同理，当缓冲区为空时，所有消费者也均被堵塞，直到有生产者向其中放入资源。

## 2. 开发环境

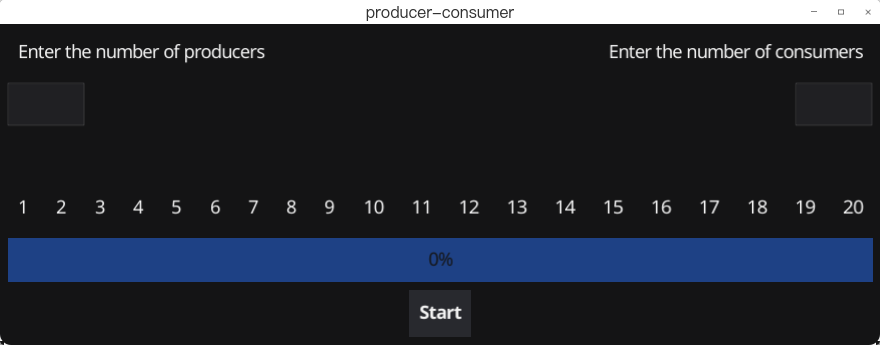
本题采用golang作为开发语言，在linux平台上进行开发，gui库使用golang 的第三方gui库[fyne.io/fyne/v2](https://pkg.go.dev/fyne.io/fyne/v2#section-readme)进行开发。

## 3.代码设计

本程序采用面向过程编程，包含一个主函数和三个子函数：begin()，producer()和consumer()。

* main函数
* main函数用来创建程序gui界面，在其中创建了一个包含输入标签、输入框、状态标签、缓冲区显示、进度条和开始按钮的布局，并将其作为窗口的内容。通过点击开始按钮，会调用begin()函数。输入框分别输入生产者和消费者的数量，状态标签用于展示当前是哪个进程占用缓冲区工作，进度条显示缓冲区占用情况。
* func main() {  
   myApp := app.New()  
   window := myApp.NewWindow("producer-consumer")  
   proNumStr = binding.NewString()  
   conNumStr = binding.NewString()  
   show = binding.NewString()  
   inputProLabel := widget.NewLabel("Enter the number of producers")  
   inputConLabel := widget.NewLabel("Enter the number of consumers")  
   inputLabelContainer := container.New(layout.NewHBoxLayout(), inputProLabel, layout.NewSpacer(), inputConLabel) //生产者消费者输入提示容器  
   inputProNum := widget.NewEntryWithData(proNumStr)  
   inputConNum := widget.NewEntryWithData(conNumStr)  
   inputBoxContainer := container.New(layout.NewHBoxLayout(), inputProNum, layout.NewSpacer(), inputConNum) //生产者消费者输入框容器  
   showLabel := widget.NewLabelWithData(show)  
   showLabelContainer := container.New(layout.NewHBoxLayout(), layout.NewSpacer(), showLabel, layout.NewSpacer()) //展示标签容器  
   showBufferContainer := container.New(layout.NewHBoxLayout()) //缓冲区内容展示容器哦  
   grids := make([]\*widget.Label, 20)  
   for i := 0; i < BufferSize; i++ {  
   grids[i] = widget.NewLabel(strconv.Itoa(i + 1))  
   }  
   for \_, grid := range grids {  
   showBufferContainer.Add(grid)  
   }  
   percentage = binding.NewFloat()  
   percentage.Set(float64(len(buffer)) / float64(BufferSize))  
   progressBar := widget.NewProgressBarWithData(percentage)  
   progressBarContainer := container.New(layout.NewMaxLayout(), progressBar) //进度条容器  
   startButton := widget.NewButton("Start", begin)  
   buttonContainer := container.New(layout.NewHBoxLayout(), layout.NewSpacer(), startButton, layout.NewSpacer()) //开始按钮容器  
   allContainer := container.New(layout.NewVBoxLayout(), inputLabelContainer, inputBoxContainer, showLabelContainer, showBufferContainer, progressBarContainer, buttonContainer)  
   window.SetContent(allContainer)  
   window.ShowAndRun()  
  }
* begin函数
* begin函数作为程序核心逻辑的入口，获取用户输入的生产者和消费者数量，然后启动两个goroutine分别运行producer()和consumer()函数，并等待它们完成后输出“All Done”。
* func begin() {  
   doneMutex.Lock()  
   done = false  
   doneMutex.Unlock()  
   str, err := proNumStr.Get() //获得生产者数量  
   if err != nil {  
   fmt.Fprintf(os.Stderr, "%v\n", err)  
   os.Exit(1)  
   }  
   proNum, err = strconv.Atoi(str)  
   if err != nil {  
   fmt.Fprintf(os.Stderr, "%v\n", err)  
   os.Exit(1)  
   }  
   str, err = conNumStr.Get() //获得消费者数量  
   if err != nil {  
   fmt.Fprintf(os.Stderr, "%v\n", err)  
   os.Exit(1)  
   }  
   conNum, err = strconv.Atoi(str)  
   if err != nil {  
   fmt.Fprintf(os.Stderr, "%v\n", err)  
   os.Exit(1)  
   }  
   wg.Add(2)  
   go producer(&wg, proNum) //启动生产者和消费者进程  
   go consumer(&wg, conNum)  
   wg.Wait()  
   fmt.Printf("All Done\n")  
  }
* producer函数
* 该函数中，通过for循环和goroutine模拟生产和消费的过程。在每个goroutine中，首先获取互斥锁对象，然后使用条件变量对象等待生产者或消费者的条件满足。在条件满足后，更新缓冲区、状态标签和进度条，释放互斥锁对象，并继续等待下一轮生产。
* func producer(wg \*sync.WaitGroup, proNum int) {  
   defer wg.Done()  
   //启动生产者协程  
   for i := 0; i < proNum; i++ {  
   wg.Add(1)  
   go func(i int) {  
   defer wg.Done()  
   for !done {  
   mutex.Lock() //获取锁  
   for len(buffer) == BufferSize { //缓冲区满时堵塞  
   cond.Wait()  
   }  
   time.Sleep(1 \* time.Second)  
   item := len(buffer) + 1  
   buffer = append(buffer, item)  
   show.Set(fmt.Sprintf("Producer(%d) produces item(%d)", i, item)) //设置展示标签  
   percentage.Set(float64(item) / float64(BufferSize)) //设置进度条百分比  
   cond.Signal()  
   mutex.Unlock() //释放锁  
   }  
   }(i)  
   }  
  }
* consumer函数
* 实现逻辑与producer函数相同，当缓冲区空时堵塞。
* func consumer(wg \*sync.WaitGroup, conNum int) {  
   defer wg.Done()  
   for i := 0; i < conNum; i++ {  
   wg.Add(1)  
   go func(i int) {  
   defer wg.Done()  
   for !done {  
   mutex.Lock()  
   for len(buffer) == 0 { //缓冲区空时堵塞  
   cond.Wait()  
   }  
   time.Sleep(1 \* time.Second)  
   item := len(buffer)  
   buffer = buffer[0 : item-1]  
   show.Set(fmt.Sprintf("Consumer(%d) consumes item(%d)", i, item))  
   percentage.Set(float64(item-1) / float64(BufferSize))  
   cond.Signal()  
   mutex.Unlock()  
   }  
   }(i)  
   }  
   wg.Done()  
  }

## 4. 程序展示

* 开始界面
* 
* 运行界面
* 