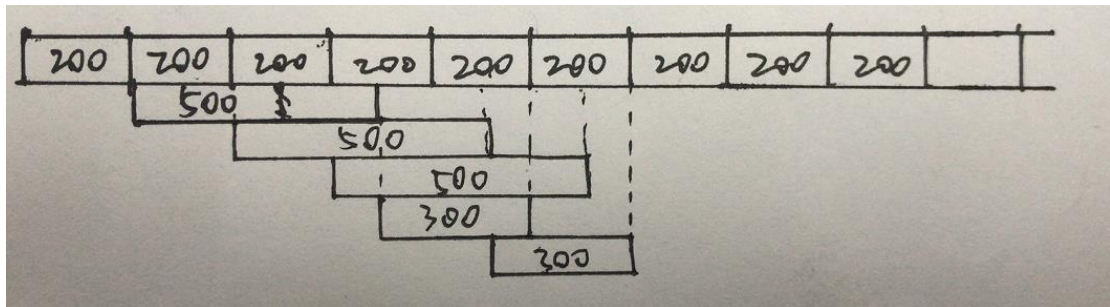


goroutine 和 channel 练习题 • 假设生成一个自然数需要 200 毫秒的时间, 对自然数 取平方需要 500 毫秒的时间, 而输出一个自然数需要 300 毫秒的时间, 请修改上一页的程序, 让输出所有数字所需时间最小。

- 提示一: 可以用 `time.Sleep(200 * time.Millisecond)` 来模拟时间消耗了 200 毫秒;
- 要求一: 在 goroutine 的阻塞时间最小同时, 请尽量减少内存花费;
- 要求二: 请说明你的程序输出每一个数字所需的时间是多少。

解决: 助教要求只有一个 counter, 所以为问题为需要多少个 squarer 和 printer 函数使得输出自然数的时间效率最快。我们从如下图片可以得到



当到第四个 200 时候, 第一个 500 执行完毕, 用于处理第四个生成的 200, 所以 500 只需要三个, 同理可得 300 只需要两个。即 squarer 和 printer 函数只需要三个和两个即可在 goroutine 的阻塞时间最小同时, 程序内存花费尽量少, 使得输出数字的效率最高。

main 执行代码为:

```
func main() {
    naturals := make(chan int)
    squares := make(chan int, 2)
    start = time.Now()

    waitGroup.Add(2)
    waitGroup2.Add(3)

    go counter(naturals)

    go squarer(squares, naturals)
    go squarer(squares, naturals)
    go squarer(squares, naturals)

    go printer2(squares)
    go printer2(squares)

    waitGroup2.Wait()
    close(squares)
    waitGroup.Wait()
    elapsed := time.Since(start)
    fmt.Println("生成全部数字的时间为: ", elapsed)
}
```

Counter、squarer、printer 代码为:

```

func counter(out chan<- int) {
    for x := 0; x < 100; x++ {
        time.Sleep(200 * time.Millisecond)
        out <- x
    }
    close(out)
}

func squarer(out chan<- int, in <-chan int) {
    for v := range in {
        time.Sleep(500 * time.Millisecond)
        out <- v * v
    }
    waitGroup2.Done()
}

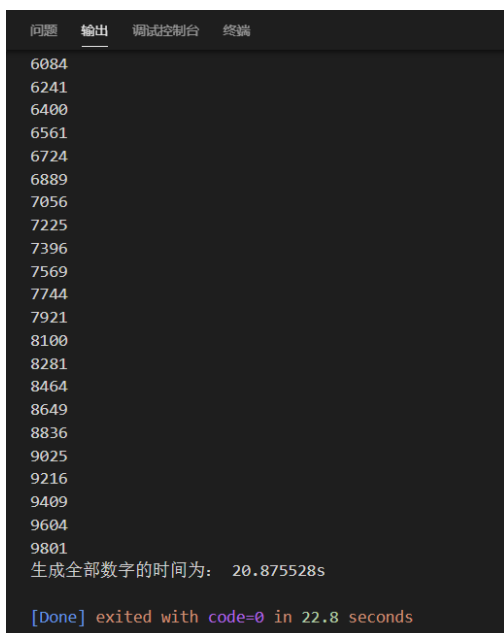
func printer(in <-chan int) {
    for v := range in {
        fmt.Println(v)
    }
    waitGroup.Done()
}

```

设置两个信号量 waitGroup, waitGroup2, 前者的作用是让 main()等待 printer 函数输出全部数字完毕才结束, 后者作用是为了 close(squares)关闭 squares 通道。

因为 counter 一个协程比较容易关闭通道, 而 squarer 有三个协程不好关闭通道, 因此我们这里设置信号量来帮忙关闭通道。

程序输出为:



```

问题 输出 调试控制台 终端
6084
6241
6400
6561
6724
6889
7056
7225
7396
7569
7744
7921
8100
8281
8464
8649
8836
9025
9216
9409
9604
9801
生成全部数字的时间为: 20.875528s

[Done] exited with code=0 in 22.8 seconds

```

符合理论值: $200 \times 100 + 500 + 300 = 20800\text{ms} = 20.8\text{s}$ 的计算结果, 而多出来的 0.07s 为程序其余的运行时间。

问题: 程序有概率出现死锁问题, 解决方案方案为对 printer 函数进行改写。

解决办法就是用 select{}避免死锁问题出现, 而 in=nil 效果可以使得 printer 函数在输出最后一个数字时候就进入 default 判断语句中, 避免输出全部数字之后, 还一直输出 0,0,0, 0,0,0...

改写代码如下:

```
//要是死锁的话可以让printer为下列函数
func printer2(in <-chan int) {
    for {
        select {
        case v, ok := <-in:
            if !ok {
                in = nil
                //在知道channel关闭后，将channel的值设为nil，这样子就相当于将这个select case子句停用了，因为nil的channel是永远阻塞的
                // fmt.Println("in is nil")
                waitGroup.Done()
                continue
            }
            time.Sleep(300 * time.Millisecond)
            fmt.Println(v)
        default:
            // 实现对死锁情况下的处理
            // time.Sleep(100 * time.Millisecond)
        }
    }
}
```

Squarer 的改写也是一样的。

```
// 死锁和关闭通道处理，没有遇到所以不采用这段代码
// func squarer2(out chan<- int, in <-chan int) {
//     for {
//         select {
//         case v, ok := <-in:
//             if !ok {
//                 waitGroup.Done()
//                 continue
//             }
//             time.Sleep(500 * time.Millisecond)
//             out <- v * v
//         default:
//             // 避免死锁
//         }
//     }
// }
// }
```

简单测试下：(死锁情况较难发生)

问题	输出	调试控制台	终端
7396			
7569			
7744			
7921			
8100			
8281			
8464			
8649			
8836			
9025			
9216			
9409			
9604			
9801			
生成全部数字的时间为: 20.8627916s			
Process exiting with code: 0			

完整源码如下：

```
package main

import (
    "fmt"
    "sync"
```

```

    "time"
)

var start = time.Now()
var waitGrouptp = sync.WaitGroup{}
var waitGroupt2 = sync.WaitGroup{}

func counter(out chan<- int) {
    for x := 0; x < 100; x++ {
        time.Sleep(200 * time.Millisecond)
        out <- x
    }
    close(out)
}

func squarer(out chan<- int, in <-chan int) {
    for v := range in {
        time.Sleep(500 * time.Millisecond)
        out <- v * v
    }
    waitGroupt2.Done()
}

// 死锁和关闭通道处理，没有遇到所以不采用这段代码
// func squarer2(out chan<- int, in <-chan int) {
//     for {
//         select {
//             case v, ok := <-in:
//                 if !ok {
//                     waitGrouptp.Done()
//                     continue
//                 }
//                 time.Sleep(500 * time.Millisecond)
//                 out <- v * v
//             default:
//                 // 避免死锁
//         }
//     }
// }

func printer(in <-chan int) {
    for v := range in {
        fmt.Println(v)
    }
}

```

```

    waitGrouptp.Done()
}

//要是死锁的话可以让 printer 为下列函数
func printer2(in <-chan int) {
    for {
        select {
            case v, ok := <-in:
                if !ok {
                    in = nil
                    //在知道 channel 关闭后，将 channel 的值设为 nil，这样子就相当于将这个 select case 子句停用了，因为 nil 的 channel 是永远阻塞的
                    // fmt.Println("in is nil")
                    waitGrouptp.Done()
                    continue
                }
                time.Sleep(300 * time.Millisecond)
                fmt.Println(v)
            default:
                // 实现对死锁情况下的处理
                // time.Sleep(100 * time.Millisecond)
        }
    }
}

func main() {
    naturals := make(chan int)
    squares := make(chan int, 2)
    start = time.Now()

    waitGrouptp.Add(2)
    waitGrouptp2.Add(3)

    go counter(naturals)

    go squarer(squares, naturals)
    go squarer(squares, naturals)
    go squarer(squares, naturals)

    go printer2(squares)
    go printer2(squares)

    waitGrouptp2.Wait()
    close(squares)
}

```

```
waitGroup.Wait()
elapsed := time.Since(start)
fmt.Println("生成全部数字的时间为: ", elapsed)
}
```