银行柜员服务问题

一、实验环境

使用go语言实现，版本为go1.24.3 windows/amd64，在windows11环境下进行实验。

二、实验目的

1. 通过对进程间通信同步/互斥问题的编程实现，加深理解信号量和 P、V 操作的原理；
2. 对 Windows 或 Linux 涉及的几种互斥、同步机制有更进一步的了解；
3. 熟悉 Windows 或 Linux 中定义的与互斥、同步有关的函数。

三、实验原理

代码使用了Go语言的并发原语实现了P、V操作的同步机制：

1. **互斥锁 (mutex)**: 实现了对共享资源（队列、顾客号码）的互斥访问
2. **条件变量 (cond)**: 实现了线程间的通信，当队列非空时唤醒等待的柜员
3. 信号量模拟:
    - 顾客加入队列并调用`Signal()`相当于P操作
    - 柜员等待条件变量相当于V操作

四、项目结构

customer.go：顾客结构体与顾客队列

workers.go：顾客处理与柜员线程

main.go：I/O入口，项目入口

1.定义顾客与顾客队列结构体：

顾客：

```
type Customer struct {
    CustomerID  int  // 顾客ID
    ArrivalTime int  // 到达时间
    ServiceTime int  // 服务所需时间
    StartTime   int  // 开始服务时间
    EndTime     int  // 结束服务时间
    ClerkID     int  // 服务柜员ID
}
```

顾客队列：

```
type CustomerQueue struct {
    customers []*Customer  // 存储顾客的队列
    mutex     sync.Mutex   // 互斥锁，保护队列的并发访问
    cond      *sync.Cond   // 条件变量，用于线程间通信
}
```

2.顾客处理与柜员处理算法

顾客处理：

```
func startCustomerWorkers(customers []Customer, q *CustomerQueue) *sync.WaitGroup {
    var customerWg sync.WaitGroup
    customerWg.Add(len(customers))
    startTime := time.Now()

    for i := range customers {
        c := customers[i]
        go func(c Customer) {
            defer customerWg.Done()

            // 模拟顾客在指定时间到达
            elapsed := time.Since(startTime)
            waitDuration := time.Duration(c.ArrivalTime)*time.Second - elapsed
            if waitDuration > 0 {
                time.Sleep(waitDuration)
            }

            // 取号（号码递增）
            numberMutex.Lock()
            currentNumber++
            numberMutex.Unlock()

            // 创建顾客并加入队列等待服务
            customer := &Customer{
                CustomerID:  c.CustomerID,
                ArrivalTime: c.ArrivalTime,
```

```
                ServiceTime: c.ServiceTime,
            }

            q.mutex.Lock()
            q.customers = append(q.customers, customer)
            q.cond.Signal()  // 通知等待的柜员有新顾客
            q.mutex.Unlock()
        }(c)
    }
    return &customerWg
}
```

柜员处理：

```
func startClerkWorkers(n int, q *CustomerQueue, results chan *Customer) *sync.WaitGroup {
    var clerkWg sync.WaitGroup
    clerkWg.Add(n)
    for i := 0; i < n; i++ {
        go func(id int) {
            defer clerkWg.Done()
            clerkAvailableTime := 0
            for {
                q.mutex.Lock()
                for len(q.customers) == 0 {
                    q.cond.Wait()  // 无顾客时等待
                }
                customer := q.customers[0]
                q.customers = q.customers[1:]
                q.mutex.Unlock()

                if customer == nil {
                    return  // 结束信号
                }

                // 处理顾客
                startTime := max(customer.ArrivalTime, clerkAvailableTime)
                endTime := startTime + customer.ServiceTime
                clerkAvailableTime = endTime

                customer.StartTime = startTime
                customer.EndTime = endTime
                customer.ClerkID = id
                results <- customer  // 发送处理结果
            }
        }(i)
    }
    return &clerkWg
}
```

3.实现要求分析

1. 使用了互斥锁`numberMutex`来保护`currentNumber`的增加操作，确保每次只有一个顾客能取到号码：

```
numberMutex.Lock()
currentNumber++
numberMutex.Unlock()
```

2. 队列操作使用互斥锁`q.mutex`保护，确保只有一个柜员能从队列中取出顾客：

```
q.mutex.Lock()
customer := q.customers[0]
q.customers = q.customers[1:]
q.mutex.Unlock()
```

3. 柜员使用条件变量等待队列中有顾客：

```
for len(q.customers) == 0 {
    q.cond.Wait()  // 无顾客时等待
}
```

4. 通过队列机制实现顾客加入队列后，必须等待前面的顾客被柜员处理完毕。

5. 当顾客到达并加入队列时，会通知等待的柜员：

```
q.mutex.Lock()
q.customers = append(q.customers, customer)
q.cond.Signal()  // 通知等待的柜员
q.mutex.Unlock()
```

五、项目运行

切换到 lab1/src 目录下，执行`go run . <clerk_num>`来启动项目，随后根据提示输入测试数据：

```
(base) D:\CODE\THUEE-OS-lab [master +1 ~2 -0 !]> cd lab1/src
(base) D:\CODE\THUEE-OS-lab\lab1\src [master +1 ~2 -0 !]> go run . 3
请输入顾客信息，每行格式为：顾客序号 进入银行时间 服务时间，输入空行结束：
1 1 10
2 5 2
3 6 3
4 10 10
5 11 1
6 15 6
7 20 6
8 22 5

顾客id: 1 到达时间: 1 开始服务时间: 1 结束服务时间: 11 服务柜员id: 1
顾客id: 2 到达时间: 5 开始服务时间: 5 结束服务时间: 7 服务柜员id: 0
顾客id: 3 到达时间: 6 开始服务时间: 6 结束服务时间: 9 服务柜员id: 2
顾客id: 4 到达时间: 10 开始服务时间: 11 结束服务时间: 21 服务柜员id: 1
顾客id: 5 到达时间: 11 开始服务时间: 11 结束服务时间: 12 服务柜员id: 0
顾客id: 6 到达时间: 15 开始服务时间: 15 结束服务时间: 21 服务柜员id: 2
顾客id: 7 到达时间: 20 开始服务时间: 21 结束服务时间: 27 服务柜员id: 1
顾客id: 8 到达时间: 22 开始服务时间: 27 结束服务时间: 32 服务柜员id: 1
(base) D:\CODE\THUEE-OS-lab\lab1\src [master +1 ~2 -0 !]> go run . 1
请输入顾客信息，每行格式为：顾客序号 进入银行时间 服务时间，输入空行结束：
1 1 10
2 5 2
3 6 3

顾客id: 1 到达时间: 1 开始服务时间: 1 结束服务时间: 11 服务柜员id: 0
顾客id: 2 到达时间: 5 开始服务时间: 11 结束服务时间: 13 服务柜员id: 0
顾客id: 3 到达时间: 6 开始服务时间: 13 结束服务时间: 16 服务柜员id: 0
(base) D:\CODE\THUEE-OS-lab\lab1\src [master +1 ~2 -0 !]>
```

六、思考题解答

1.柜员人数和顾客人数对结果分别有什么影响?

顾客人数越多，所用时间越多，大体呈线性趋势；随着柜员人数，所用时间先快速减少，随后趋于平稳。

2.实现互斥的方法有哪些?各自有什么特点?效率如何?

1. 忙等待：不断判断某些变量的值。实现简单，无需进入内核态。缺点是占用CPU资源且效率较低，还可能出现优先级反转问题。
2. 信号量：采用P、V原语实现互斥与同步。CPU使用效率较高且阻塞不会占用CPU资源。缺点是用户态与内核态之间的切换会带来CPU资源的开销。
3. 互斥锁：效率相比信号量更高，但本质上是一种特殊简化的信号量，与信号量有同样的问题。
4. 管程：只允许一个线程在管程函数内，互斥由编译器来实现。