НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та
спеціалізованих комп‘ютерних систем

Лабораторна робота №2

З дисципліни: «Системне програмне забезпечення»

«Інженерія програмного забезпечення (ОС)»

Студент КВ-42**:** Савицький Т.П.
Перевірив(ла):
_____
_____

Київ-2017

Завдання

Автомат для розміну грошей (монет)

Автомат приймає монету та ідентифікує її (визначає номінал). Ідентифікація моделюється датчиком випадкових чисел. Приймаються монети номіналом 1, 2, 5, 10, 25, 50 коп. і 1 грн. Розмін монети здійснюється на монети номіналу, що вводиться з клавіатури.

Кількість монет різного номіналу (1, 2, 5, 10, 25, 50 коп.), що містить автомат для розміну та видачі здачі, наперед задається. Якщо розмін и видача монет можливі, вони виконуються. Інакше здійснюється відмова від виконання операції розміну і видається повідомлення про її причину. Монети, що приймаються для розміну, додаються до наявних в автоматі, та, у свою чергу, можуть використовуватися для розміну. Модель роботи автомату представити у вигляді двох взаємодіючих процесів A і B: процес A визначає момент надходження монети та ідентифікує її, а процес B здійснює розмін і видає гроші або відмову від виконання операції розміну.

Для організації доступу до подільних ресурсів використати *семафори*

Семафор — це універсальний механізм для організації взаємодії процесів. Розв'язує задачі взаємного виключення та синхронізації потоків. Він є одним з найстаріших засобів розподілення доступу процесів, що працюють паралельно, до критичних ресурсів. Семафори використовуються для контролю доступу до спільного ресурсу, або для синхронізації процесів (потоків).

Текст програми

```
package semaphore

import (
	"sync"
)

// Semaphore is an implementation of semaphore.
type Semaphore struct {
	permits int
	avail   int
	channel chan struct{}
	aMutex  sync.Mutex   // acquire
	rMutex  sync.Mutex   // release
	pMutex  sync.RWMutex // number of permits
}

// New creates a new Semaphore with specified number of permits.
func New(permits int) *Semaphore {
	if permits < 1 {
		panic("Invalid number of permits. Less than 1")
	}

	// fill channel buffer
	channel := make(chan struct{}, permits)
	for i := 0; i < permits; i++ {
		channel <- struct{}{}
	}

	return &Semaphore{
		permits: permits,
		avail:   permits,
		channel: channel,
	}
}

// Acquire acquires one permit. If it is not available, the goroutine will block until it is available.
func (s *Semaphore) Acquire() {
	s.aMutex.Lock()
```

```
		defer s.aMutex.Unlock()

		s.pMutex.Lock()
		s.avail--
		s.pMutex.Unlock()


		<-s.channel
}

// AcquireMany is similar to Acquire() but for many permits.
//
// The number of permits acquired is at most the number of permits in the semaphore.
// i.e. if n = 5 and s was created with New(2), at most 2 permits will be acquired.
func (s *Semaphore) AcquireMany(n int) {
		if n > s.permits {
			n = s.permits
		}

		for ; n > 0; n-- {
			s.Acquire()
		}
}

// Release releases one permit.
func (s *Semaphore) Release() {
		s.rMutex.Lock()
		defer s.rMutex.Unlock()

		s.channel <- struct{}{}

		s.pMutex.Lock()
		s.avail++
		s.pMutex.Unlock()
}

// ReleaseMany releases n permits.
//
// The number of permits released is at most the number of permits in the semaphore.
// i.e. if n = 5 and s was created with New(2), at most 2 permits will be released.
func (s *Semaphore) ReleaseMany(n int) {
		if n > s.permits {
			n = s.permits
		}

		for ; n > 0; n-- {
			s.Release()
		}
}

// AvailablePermits gives number of available unacquired permits.
func (s *Semaphore) AvailablePermits() int {
		s.pMutex.RLock()
		defer s.pMutex.RUnlock()

		if s.avail < 0 {
			return 0
		}
		return s.avail
}


package bancomat

import (
		"math/rand"
		"../semaphore"
		"sync"
)

var initStorage []int
var box = Bancomat{}
var g = &sync.WaitGroup{}

func InitValue() {
		initStorage = []int{1,2,2,10,10,25,25,50,50,50,100}
		box.semResult = semaphore.New(1)
		box.nominal = map[string]int{
			"1"		: 1,
			"2"		: 2,
			"5"		: 5,
```

```
                "10"    : 10,
                "25"    : 25,
                "50"    : 50,
                "100"   : 100,
        }
}


type Bancomat struct {
        result []interface{}
        nominal map[string]int
        semResult *semaphore.Semaphore
}

func Exchange(value int, returnNominal []int) []interface{} {
        nominal := identifierNominal(value)
        g.Add(1)
        go exchangeProcess(nominal, returnNominal)
        g.Wait()
        result := result()
        return result
}

func exchangeProcess(value int, returnNominal []int) {
        counter := 0
        isContinue := false

        for _, v := range initStorage {
                if contains(returnNominal, v) {
                        if (counter + v) > value {
                                continue
                        }
                        counter += v
                        if counter == value {
                                isContinue = true
                                break
                        }
                }
        }


        if isContinue {
                var s []int
                m := initStorage
                deleted := 0
                counter := 0
                for i := range m {
                        j := i - deleted
                        if contains(returnNominal, m[j]) {
                                s = append(s, m[j])
                                counter += m[j]
                                m = m[:j+copy(m[j:], m[j+1:])]
                                deleted++
                                if counter == value {
                                        isContinue = true
                                        break
                                }
                        }
                }
                setResult([]interface{}{
                        "Success",
                        s,
                })
        } else {
                setResult([]interface{}{
                        "Error",
                        []int{},
                })
        }

        g.Done()
}

func contains(s []int, e int) bool {
        for _, a := range s {
                if a == e {
                        return true
                }
        }
        return false
}
```

```
func result() []interface{} {
	box.semResult.Acquire()
	defer box.semResult.Release()
	return box.result
}

func setResult(r []interface{}) {
	box.semResult.Acquire()
	box.result = r
	box.semResult.Release()
}


func identifierNominal(v int) int {
	for i := 0; i < 100000; i++ {
		r := randIntMapKey(box.nominal)
		if r == v {
			return v
		}
	}
	panic("Don't identifier nominal")
}

func randIntMapKey(m map[string]int) int {
	i := rand.Intn(len(m))
	for _, v := range m {
		if i == 0 {
			return v
		}
		i--
	}
	panic("Error in rand map key")
}
```

```
package main

import (
	"fmt"
	"./bancomat"
)

func main() {
	bancomat.InitValue()
	printResult(bancomat.Exchange(100, []int{25,50}))
	printResult(bancomat.Exchange(100, []int{50}))
	printResult(bancomat.Exchange(25, []int{1,2,10}))
	printResult(bancomat.Exchange(50, []int{25}))
}

func printResult(r []interface{}) {
	fmt.Println(r[0], ":", r[1])
}
```

Тести