Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени Н.Э.БАУМАНА
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №4

«Задача о пяти обедающих философах.»

Выполнил
студент группы ИУ9-51Б
Лисов Алексей

Москва, 2024

1 Условие

Суть задачи следующая. Пять философов сидят за круглым столом. Они проводят жизнь, чередуя приёмы пищи и размышления. В центра стола находится большое блюдо спагетти. Чтобы съесть порцию, каждому философу нужно две вилки. Однако, вилок всего пять: между каждой парой рядом сидящих философов лежат по одной вилке, и каждый философ может пользоваться только теми вилками, которые лежат рядом с ним, слева и справа. Философ не может брать две вилки одновременно: сначала он тратит некоторое время на то, чтобы взять одну, затем вторую. Однако, он может одновременно положить их на место. Задача заключается в том, чтобы написать программу, моделирующую поведение философов. Очевидно, что раз вилок всего пять, то одновременно есть могут не более двух философов, и два сидящих рядом философа не могут есть одновременно. Для имитации периодов раздумий и приёмов пищи можно использовать генератор случайных чисел, позволяющий задавать времена их действий в определённом интервале. Имитация поведения каждого философа, по сути, разбивается на то, что в любой момент времени философ находится в одном из пяти состояний: размышляет, берёт левую вилку, берёт правую вилку, ест, кладёт вилки на место. Таким образом, вилки являются разделяемым ресурсом. На программу накладываются условия: 1. Каждый философ, по сути, является потоком, и модель поведения у каждого из них должна быть одинаковой, кроме того, какие вилки они могут брать. 2. Накладывание блокировки по сути является действием по взятию вилки, поэтому накладывать блокировку сразу на обе вилки нельзя; последовательность действий должна быть «наложить блокировку – взять вилку – наложить вторую блокировку – взять вторую вилку». 3. Программа должна избегать ситуации взаимоблокировки: ситуации, в которой все философы голодны, то есть ни один из них не может взять себе две вилки (например, когда каждый держит по одной и не хочет её отдавать). Запрограммировать остановку алгоритма по достижении контрольного времени (например, атомарной операцией над булевым флагом). В отчёте построить некоторый результат работы алгоритма, которая может быть в виде графика, таблицы, лога или чего угодно ещё; главное условие состоит в том, чтобы по результатам можно было однозначно определить, чем в каждый момент времени был занят каждый философ (одно из пяти состояний). Также рассмотреть вариант программы с увеличением количества философов до произвольного N.

2 Код решения

Содержимое Go файла:

```go
//Имя Go файла: main.go
//Содержимое Go файла:

package main

import (
        "context"
        app2 "github.com/BaldiSlayer/rprp/lab4/internal/app"
        "log"
        "time"
)

var (
        ctx = context.Background()
)

func main() {
        app := app2.New()

        ctx2, cancel := context.WithTimeout(ctx, 40*time.Second)
        defer cancel()

        err := app.Run(ctx2, 5)
        if err != nil {
                log.Printf("some philosophers are hungry :(\nreason:%v", err)
        }

        log.Print("end of philosophers lives")
}

----------------------

//Имя Go файла: fork.go
//Содержимое Go файла:
package fork

import (
        "math/rand"
        "sync"
        "time"
```

```
)

func getMillis(a, b int64) time.Duration {
        randomMcs := rand.Int63n(b-a+1) + a

        return time.Duration(randomMcs) * time.Millisecond
}

type Fork struct {
        fork *sync.Mutex
}

func New(f *sync.Mutex) Fork {
        return Fork{
                fork: f,
        }
}

func (f *Fork) Get() bool {
        if f.fork.TryLock() {
                // мы не можем взять вилку мгновенно
                time.Sleep(getMillis(100, 1000))

                return true
        }

        return false
}

func (f *Fork) Release() {
        f.fork.Unlock()
}

----------------------

//Имя Go файла: app.go
//Содержимое Go файла:
package app

import (
```

```
        "context"
        "github.com/BaldiSlayer/rprp/lab4/internal/fork"
        "github.com/BaldiSlayer/rprp/lab4/internal/phr"
        "sync"
)

type App struct{}

func New() *App {
        return &App{}
}

func philosophersLive(ctx context.Context, phs []phr.Philosopher) chan struct{} {
        ch := make(chan struct{})

        go func() {
                var wg sync.WaitGroup

                wg.Add(len(phs))

                for _, philosopher := range phs {
                        philosopher := philosopher

                        go func() {
                                defer wg.Done()

                                philosopher.Live(ctx)
                        }()
                }

                wg.Wait()

                ch <- struct{}{}
        }()

        return ch
}

func (app *App) Run(ctx context.Context, phNum int) error {
        forks := make([]fork.Fork, phNum)
```

```
        for i := 0; i < phNum; i++ {
                forks[i] = fork.New(&sync.Mutex{})
        }

        phs := make([]phr.Philosopher, phNum)

        for i := 0; i < phNum; i++ {
                phs[i] = phr.New(
                        i,
                        forks[i],
                        forks[(i+1)%phNum],
                )
        }

        select {
        case <-philosophersLive(ctx, phs):
        case <-ctx.Done():
                return ctx.Err()
        }

        return nil
}

----------------------

//Имя Go файла: defaults.go
//Содержимое Go файла:
package defaults

const (
        MaxRandTime         = 1228
        MaxForkGetRandTime  = 1228
        PhilosopherWaitTime = 300
        PhilosopherCooldown = 5000
)

----------------------

//Имя Go файла: philosopher.go
```

```go
//Содержимое Go файла:
// Package phr - philosopher package
package phr

import (
        "context"
        "github.com/BaldiSlayer/rprp/lab4/internal/fork"
        "log"
        "math/rand"
        "time"
)

func getMillis(a, b int64) time.Duration {
        randomMcs := rand.Int63n(b-a+1) + a

        return time.Duration(randomMcs) * time.Millisecond
}

type Philosopher struct {
        id        int
        leftFork  fork.Fork
        rightFork fork.Fork
}

func New(id int, leftFork, rightFork fork.Fork) Philosopher {
        return Philosopher{
                id:        id,
                leftFork:  leftFork,
                rightFork: rightFork,
        }
}

func (p *Philosopher) think() {
        log.Printf("Philosopher %d is thinking\n", p.id)
        time.Sleep(getMillis(100, 1000))
}

func (p *Philosopher) eat() {
        log.Printf("Philosopher %d is eating\n", p.id)
        time.Sleep(getMillis(100, 1000))
```

```
        log.Printf("Philosopher %d ended eating\n", p.id)
}

func (p *Philosopher) getForksIter() bool {
        if ok := p.leftFork.Get(); !ok {
                return false
        }

        if ok := p.rightFork.Get(); !ok {
                p.leftFork.Release()

                return false
        }

        return true
}

func (p *Philosopher) getForks() {
        for {
                if ok := p.getForksIter(); ok {
                        return
                }

                time.Sleep(getMillis(100, 500))
        }
}

func (p *Philosopher) putAllForks() {
        p.leftFork.Release()
        p.rightFork.Release()
}

func (p *Philosopher) Live(ctx context.Context) {
        p.think()
        p.getForks()
        p.eat()
        p.putAllForks()
}

----------------------
```

3 Лог

```
2024/12/14 02:44:50 Philosopher 4 is thinking
2024/12/14 02:44:50 Philosopher 1 is thinking
2024/12/14 02:44:50 Philosopher 0 is thinking
2024/12/14 02:44:50 Philosopher 3 is thinking
2024/12/14 02:44:50 Philosopher 2 is thinking
2024/12/14 02:44:50 Philosopher 1 is eating
2024/12/14 02:44:51 Philosopher 1 ended eating
2024/12/14 02:44:52 Philosopher 4 is eating
2024/12/14 02:44:52 Philosopher 2 is eating
2024/12/14 02:44:52 Philosopher 4 ended eating
2024/12/14 02:44:52 Philosopher 2 ended eating
2024/12/14 02:44:53 Philosopher 0 is eating
2024/12/14 02:44:54 Philosopher 3 is eating
2024/12/14 02:44:54 Philosopher 0 ended eating
2024/12/14 02:44:55 Philosopher 3 ended eating
2024/12/14 02:44:55 end of philosophers lives
```

Характеристики устройства

- Процессор - Apple M2 Pro
- Оперативная память - 16 Гб
- Операционная система - MacOS Sonoma 14.5