НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт з комп'ютерного практикуму №2

«Об'єктно-орієнтований підхід до побудови імітаційних моделей дискретно-подійних систем.»

роботи з дисципліни: « Моделювання систем »

Студент: Мєшков Андрій Ігорович

Група: ІП-15

Викладач: асистент Дифучин А. Ю.

Київ, 2024

Завдання

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об'єктно-орієнтованого підходу. **5 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою. **5 балів.**
3. Створити модель за схемою, представленою на рисунку 2.1. **30 балів.**
4. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці. **10 балів.**

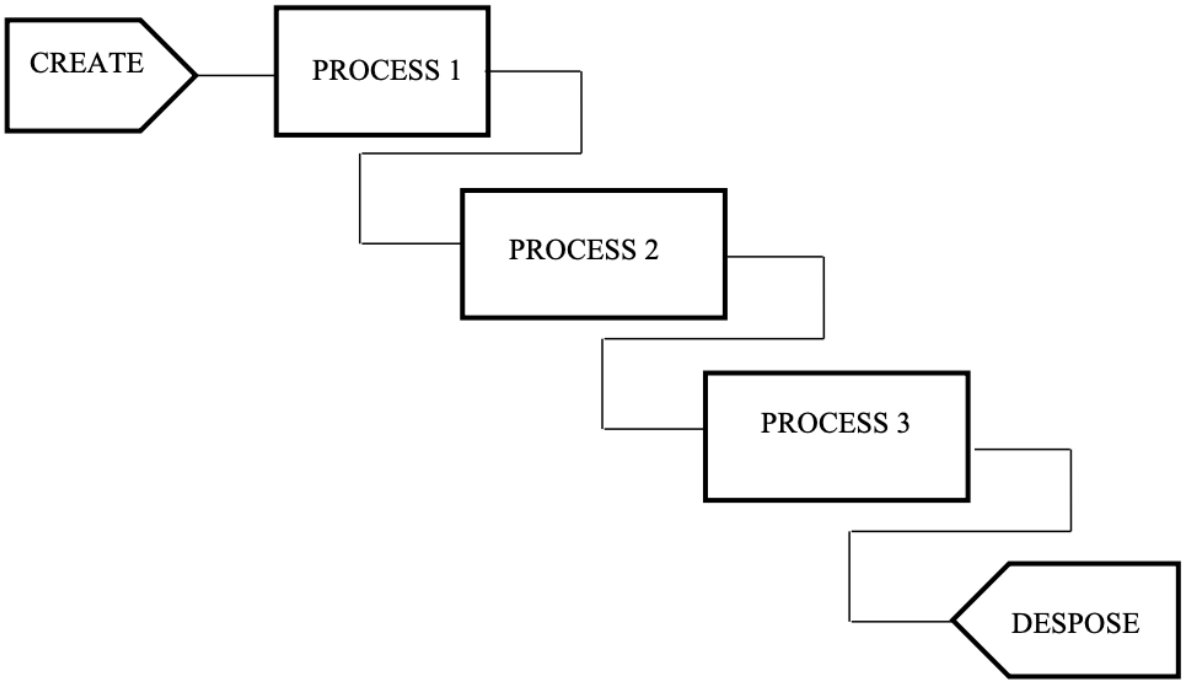

Рисунок 2.1 – Схема моделі.

5. Модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями. **20 балів.**
6. Модифікувати клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у попередні блоки. **30 балів.**

Хід роботи

Було створено модель.

```
import Foundation

let create = Create(name: "create", delay: 1)
let process1 = Process(name: "process1", delays: [5, 5, 5, 5, 5])
let process2 = Process(name: "process2", delays: [1.8, 1.8])
let process3 = Process(name: "process3", delays: [1])

create.transfer = SoloTransfer(nextElement: process1)
process1.transfer = SoloTransfer(nextElement: process2)
process2.transfer = SoloTransfer(nextElement: process3)

process1.maxQueue = 1
process2.maxQueue = 2
process3.maxQueue = 5

create.distribution = .exp
process1.distribution = .exp
process2.distribution = .exp
process3.distribution = .exp

 let model = Model(elements: [create, process1, process2, process3])
model.simulate(timeModeling: 1000)
```

```
-------------RESULTS-------------

Result of work create
quantity = 1016

Result of work process1
quantity = 767
failure probability = 0.24209486166007904
mean length of queue = 0.2366794191653162
mean working devices = 3.784182017354707

Result of work process2
quantity = 707
failure probability = 0.08062418725617686
mean length of queue = 0.31710191698619705
mean working devices = 1.2277734558901563

Result of work process3
quantity = 679
failure probability = 0.03551136363636364
mean length of queue = 1.0918170134604634
mean working devices = 0.7081542648879964
```

Рисунок 2.1. Результат імітації

Перевіримо працездатність моделі:

Верифікація ММО																									
	Значення вхідних змінних											Значення вихідних змінних моделі													
Прогон	Середній інтервал надходження вимог	Кількість пристроїв СМО1	Обмеження на довжину черги СМО1	Середня тривалість обслуговування СМО1	Кількість пристроїв СМО2	Обмеження на довжину черги СМО2	Середня тривалість обслуговування СМО2	Кількість пристроїв СМО3	Обмеження на довжину черги СМО3	Середня тривалість обслуговування СМО3	Кількість вимог, що надійшли в мережу	Кількість обслугованих в СМО1	Ймовірність відмови СМО1	Середня довжина черги в СМО1	Середня кількість зайнятих пристрої в СМО1	Кількість обслугованих в СМО2	Ймовірність відмови СМО2	Середня довжина черги в СМО2	Середня кількість зайнятих пристроїв СМО2	Кількість обслугованих в СМО3	Ймовірність відмови СМО3	Середня довжина черги в СМО3	Середня кількість зайнятих пристроїв СМО3
(t=1000)	2,00	3,00	5,00	1,50	1,00	5,00	1,80	1,00	5,00	1,00	530,000	530,000	0,000	0,012	0,825	465,000	0,123	2,182	0,880	464,000	0,002	0,339	0,422
(t=1000)	1,00	3,00	5,00	1,50	1,00	5,00	1,80	1,00	5,00	1,00	980,000	977,000	0,004	0,183	1,391	553,000	0,432	3,811	0,985	548,000	0,009	0,676	0,574
(t=1000)	1,00	5,00	5,00	1,50	1,00	5,00	1,80	1,00	5,00	1,00	993,000	993,000	0,000	0,001	1,426	528,000	0,467	3,933	0,987	523,000	0,008	0,397	0,468
(t=1000)	1,00	5,00	1,00	1,50	1,00	5,00	1,80	1,00	5,00	1,00	989,000	987,000	0,003	0,003	1,452	521,000	0,471	3,936	0,989	521,000	0,000	0,367	0,499
(t=1000)	1,00	5,00	1,00	5,00	1,00	5,00	1,80	1,00	5,00	1,00	1014,000	774,000	0,234	0,226	3,920	568,000	0,265	3,090	0,938	558,000	0,012	0,759	0,590
(t=1000)	1,00	5,00	1,00	5,00	2,00	5,00	1,80	1,00	5,00	1,00	1047,000	805,000	0,229	0,236	3,942	784,000	0,022	0,888	1,414	747,000	0,048	1,458	0,750
(t=1000)	1,00	5,00	1,00	5,00	2,00	2,00	1,80	1,00	5,00	1,00	1016,000	767,000	0,242	0,237	3,784	707,000	0,081	0,317	1,228	679,000	0,036	1,092	0,708

Як бачимо, модель відповідає всім правилам, а отже пройшла верифікацію.

Модифікуємо програму.

```
import Foundation

let create = Create(name: "create", delay: 1)
let process1 = Process(name: "process1", delays: [5, 5, 5, 5, 5])
let process2 = Process(name: "process2", delays: [1.8, 1.8])
let process3 = Process(name: "process3", delays: [1])
let process4 = Process(name: "process4", delays: [0.1])

create.transfer = SoloTransfer(nextElement: process1)
process1.transfer = SoloTransfer(nextElement: process2)
process2.transfer = TranferWithProbability(probabilities: [
    TransferProbability(probability: 0.5, nextElement: process1),
    TransferProbability(probability: 0.5, nextElement: process3),
    TransferProbability(probability: 0.5, nextElement: process4)
])

process1.maxQueue = 1
process2.maxQueue = 2
process3.maxQueue = 5
process4.maxQueue = 1

create.distribution = .exp
process1.distribution = .exp
process2.distribution = .exp
process3.distribution = .exp
process4.distribution = .exp

let model = Model(elements: [create, process1, process2, process3, process4])
model.simulate(timeModeling: 1000)
```

ВИСНОВКИ

У результаті виконання практичної роботи було розроблено модель масового обслуговування, що підтримує роботу декількох пристроїв та переходи з ймовірностями. Було проведено верифікацію моделі

ЛІСТИНГ КОДУ

--- main.swift ---

```
import Foundation

let create = Create(name: "create", delay: 1)
let process1 = Process(name: "process1", delays: [5, 5, 5, 5, 5])
let process2 = Process(name: "process2", delays: [1.8, 1.8])
let process3 = Process(name: "process3", delays: [1])
let process4 = Process(name: "process4", delays: [0.1])

create.transfer = SoloTransfer(nextElement: process1)
process1.transfer = SoloTransfer(nextElement: process2)
//process2.transfer = SoloTransfer(nextElement: process3)
process2.transfer = TranferWithProbability(probabilities: [
    TransferProbability(probability: 0.5, nextElement: process1),
    TransferProbability(probability: 0.5, nextElement: process3),
    TransferProbability(probability: 0.5, nextElement: process4)
])

process1.maxQueue = 1
process2.maxQueue = 2
process3.maxQueue = 5
process4.maxQueue = 1

create.distribution = .exp
process1.distribution = .exp
process2.distribution = .exp
process3.distribution = .exp
process4.distribution = .exp

// let model = Model(elements: [create, process1, process2, process3])
let model = Model(elements: [create, process1, process2, process3, process4])
model.simulate(timeModeling: 1000)
```

--- Element/Process.swift ---

```
import Foundation

class Process: Element {

    private var queue = 0
```

```
var maxQueue = Int.max
private(set) var failure = 0
private(set) var meanQueue = 0.0
private(set) var loadTime = 0.0
private(set) var workingDevicesCount = 0.0

init(name: String, delays: [Double]) {
    super.init(nameOfElement: name, delays: delays)
}

override func inAct() {
    switch state {
    case .free:
        devices.first(where: {$0.state == .free})?.inAct()
    case .working:
        if queue < maxQueue {
            queue += 1
        } else {
            failure += 1
        }
    }
}

override func outAct() {
    super.outAct()
    let outCount = devices.filter({ $0.tNext == tCurr }).count
    devices.filter({ $0.tNext == tCurr }).forEach({ $0.outAct() })
    (0..<outCount).forEach({ _ in
        if queue > 0 {
            queue -= 1
            devices.first(where: {$0.state == .free})?.inAct()
        } else {
            devices.filter({ $0.state == .free }).forEach({ $0.tNext = .infinity })
        }
        transfer?.goNext()
    })
}

override func printInfo() {
    super.printInfo()
    print("failure = \(failure)")
}
```

```
    override func doStatisctic(delta: Double) {
        meanQueue += Double(queue) * delta
        if devices.filter({ $0.state == .working }).count > 0 {
            loadTime += delta
        }
        workingDevicesCount += Double(devices.filter({ $0.state == .working
}).count) * delta
    }

}


--- Element/Element.swift ---
import Foundation

class Element {

    private static var nextID = 0

    let name: String
    var distribution = Method.exp
    private(set) var quantity = 0
    var tCurr = 0.0 {
        didSet {
            devices.forEach({ $0.tCurr = tCurr })
        }
    }
    var transfer: Transfer?
    let id: Int

    var devices: [Device]

    init(nameOfElement: String, delays: [Double]) {
        id = Element.nextID
        Element.nextID += 1
        name = nameOfElement
        devices = []
        delays.forEach({ devices.append(Device(delay: $0, distribution:
distribution)) })
    }

    var tNext: Double {
        devices.map({ $0.tNext }).min()!
```

```
    }

    var state: State {
        if devices.filter({ $0.state == .free }).count > 0 {
            return .free
        } else {
            return .working
        }
    }

    func inAct() {

    }

    func outAct() {
        quantity += 1
    }

    func printResult() {
        print("Result of work \(name) \nquantity = \(quantity)")
    }

    func printInfo() {
        print("\(name) state = \(state.rawValue) quantity = \(quantity) tNext = \(tNext)")
    }

    func doStatisctic(delta: Double) {

    }
}

enum Method {
    case exp, norm, unif
}

enum State: String {
    case working, free
}
```

--- Element/Create.swift ---

```
import Foundation

class Create: Element {

    init(name: String, delay: Double) {
        super.init(nameOfElement: name, delays: [delay])
        devices[0].tNext = 0
    }

    override func outAct() {
        super.outAct()
        devices[0].outAct()
        devices[0].tNext += devices[0].delay
        transfer?.goNext()
    }

}


--- Transfer/SoloTransfer.swift ---
import Foundation

class SoloTransfer: Transfer {

    let nextElement: Element

    init(nextElement: Element) {
        self.nextElement = nextElement
    }

    func goNext() {
        nextElement.inAct()
    }

}


--- Transfer/TranferWithProbability.swift ---

import Foundation

class TranferWithProbability: Transfer {
```

```
    let probabilities: [TransferProbability]
    let maxProbability: Double

    init(probabilities: [TransferProbability]) {
        self.probabilities = probabilities
        maxProbability = probabilities.reduce(0.0, { $0 + $1.probability })
    }

    func goNext() {
        let number = Double.random(in: 0..<maxProbability)
        var currentProbability = 0.0
        for probability in probabilities {
            currentProbability += probability.probability
            if currentProbability > number {
                probability.nextElement.inAct()
                break
            }
        }
    }

}

struct TransferProbability {
    let probability: Double
    let nextElement: Element
}


--- Transfer/Transfer.swift ---

import Foundation

protocol Transfer {
    func goNext()
}


--- FunRand/FunRand.swift ---

import Foundation

class FunRand {
```

```
    private class var generateA: Double {
        var a = Double.random(in: 0..<1)
        while a == 0 {
            a = Double.random(in: 0..<1)
        }
        return a
    }

    class func exp(timeMean: Double) -> Double {
        var a = generateA
        a = -timeMean * log(a)
        return a
    }

    class func unif(timeMin: Double, timeMax: Double) -> Double {
        var a = generateA
        a = timeMin + a * (timeMax - timeMin)
        return a
    }

    class func norm(timeMean: Double, timeDeviation: Double) -> Double {
        timeMean + timeDeviation * Double.random(in: -
Double.infinity...Double.infinity)
    }

}


--- Model/Model.swift ---
import Foundation

class Model {

    private var tNext = 0.0
    private var tCurr = 0.0
    private var event = 0
    private let elements: [Element]

    init(elements: [Element]) {
        self.elements = elements
    }

    func simulate(timeModeling: Double) {
```

```
        while tCurr < timeModeling {
            tNext = Double.infinity
            elements.forEach { element in
                if element.tNext < tNext {
                    tNext = element.tNext
                }
            }
            elements.forEach({ $0.doStatisctic(delta: tNext - tCurr) })
            tCurr = tNext
            elements.forEach({ $0.tCurr = tCurr })
            elements.forEach { element in
                if element.tNext == tCurr {
                    element.outAct()
                    print("It's time for event in \(element.name) time = \(tCurr)")
                }
            }
            elements.forEach({ $0.printInfo() })
        }
        printResult()
    }

    private func printResult() {
        print("\n------------RESULTS------------\n")
        elements.forEach { element in
            element.printResult()
            if let process = element as? Process {
                print("failure probability = \(Double(process.failure) / Double(process.quantity + process.failure))")
                print("mean length of queue = \(process.meanQueue / tCurr)")
                print("mean working devices = \(process.workingDevicesCount / tCurr)")
            }
            print("")
        }
    }

}


--- Device/Device.swift ---
import Foundation

class Device {
```

```
    var tNext = 0.0
    private let delayMean: Double
    var tCurr = 0.0
    var state = State.free
    let distribution: Method
    private let delayDev = 0.0

    var delay: Double {
        var delay = delayMean
        switch distribution {
        case .exp:
            delay = FunRand.exp(timeMean: delayMean)
        case .norm:
            delay = FunRand.norm(timeMean: delayMean, timeDeviation: delayDev)
        case .unif:
            delay = FunRand.unif(timeMin: delayMean, timeMax: delayDev)
        }
        return delay
    }

    init(delay: Double, distribution: Method) {
        self.delayMean = delay
        self.distribution = distribution
    }

    func inAct() {
        state = .working
        tNext = tCurr + delay
    }

    func outAct() {
        state = .free
    }

}
```