Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Моделирование

ОТЧЕТ

по лабораторной работе № 5

на тему

ПОСТРОЕНИЕ И ИССЛЕДОВАНИЕ ИМИТАЦИОННОЙ МОДЕЛИ НЕПРЕРЫВНО-СТОХАСТИЧЕСКОЙ СМО,

ВАРИАНТ № 5

Студенты: А.В. Гуринович

И.В. Клишевский

Проверила: Ю.О. Герман

МИНСК 2022

# 1. Цель работы

Изучить методы имитационного моделирования поведения непрерывно-стохастической СМО.

# 2. Задание

Произвести имитационное моделирование для системы с одним прибором. Интенсивность обслуживания заявок μ = 0,5 секунды. Законы распределения вероятностей времен поступления обозначены как F1(t) и F2(t), времени обслуживания – как G(t).

Законы распределения вероятностей времени поступления:

Закон распределения вероятности времени обслуживания:

По результатам моделирования найти – среднее время обслуживания, среднее время пребывания заявки в системе, среднее число заявок в системе, процент загрузки обслуживающего прибора (канала).

# 3. Ход работы

## 3.1 Нахождение времени поступления заявок и их обработки

Используем псевдослучайные числа из лабораторной работы № 3 принадлежащие промежутку [0,1;1], так как числа менее одной десятой будут давать либо слишком маленькие, либо отрицательные времена поступления. Последовательность чисел: 0.9, 0.34, 0.94, 0.12, 0.5.

Выразим общую формулу t из законов распределения вероятностей времени поступления, обозначим случайную величину как x:

Теперь можно выразить несколько первых членов последовательности, для удобства используем только первую формулу:

Теперь проведём аналогичные операции для выражения времени обслуживания из закона распределения вероятности времени обслуживания, обозначим случайную величину как x:

Однако верно будет рассчитать это не так, μ равняется двойке, а не одной второй, посчитаем правильно, однако далее по отчёту и в коде используем первоначальные неправильные вычисления, вот так…:

Теперь можно выразить несколько первых членов последовательности:

## 3.2 Пример моделирования системы

Рассмотрим работу системы, упростим используя только первую формулу распределения вероятности времени поступления заявок, в реализации же будем выбирать одну из формул случайным образом. Разделим её на этапы по времени: момент получения заявки, время нахождения заявки в очереди, обработка заявки, бездействие.

Начнём заявки № 0, она поступает в момент времени t0 = 2,59, после чего обрабатывается за время τ0 = 1,15. То есть система закончит обработку этой заявки в момент времени t0 + τ0 = 3,74. При этом момент поступления следующей заявки в систему только t0 + t1 = 4,2. То есть система будет бездействовать (t0 + t1) – (t0 + τ0) = 0,46.

Отдельно рассмотрим обслуживание заявки № 3, так как она не была поставлена на обслуживание сразу же, а попала в очередь. Сначала нужно рассмотреть выполнение заявки № 2, которое начинается в момент времени t0 + t1 + t2 = 6,83, время её выполнения составляет τ2 = 1,41, то есть обслуживание заявки закончится в момент времени t0 + t1 + t2 + τ2 = 8,24. В то же время заявка № 3 посыпает в момент времени t0 + t1 + t2 + t3 = 7,4, то есть ещё до окончания обслуживания заявки № 2. В таком случае заявку необходимо поместить в очередь, в которой она пробудет до момента окончания обработки заявки № 2, то есть (t0 + t1 + t2 + τ2) – (t0 + t1 + t2 + t3) = 0,84.

Диаграмма схемы с обозначениями вышеописанных состояний приведена на рисунке 3.1.



Рисунок 3.1 – диаграмма системы

Для программной реализации используем две формулы, выбор между которыми будет осуществляться случайно.

## 3.3 Программное решение

Общий класс для заявки:

class Entity {

var incomeTime: Double

var processingTime: Double

var processingRange: ClosedRange<Double>

var waiting: Double

var inactivity: Double

init(seed: Double, previousIncomeTime: Double, previousProcessingEndTime: Double) {

self.incomeTime = previousIncomeTime + calculateIncomeTime(seed)

self.processingTime = calculateProcessingTime(seed)

//processingRange

if incomeTime >= previousProcessingEndTime {

processingRange = incomeTime...(incomeTime + processingTime)

} else {

processingRange = previousProcessingEndTime...(previousProcessingEndTime + processingTime)

}

//waiting

if processingRange.lowerBound == incomeTime {

self.waiting = 0

} else {

self.waiting = processingRange.lowerBound - incomeTime

}

//inactivity

if previousProcessingEndTime == incomeTime, self.waiting == 0 {

self.inactivity = 0

} else {

self.inactivity = processingRange.lowerBound - previousProcessingEndTime

}

}

func print() {

Swift.print("Entity description:")

Swift.print("incomeTime: \(self.incomeTime)")

Swift.print("processingTime: \(self.processingTime)")

Swift.print("processingRange: \(self.processingRange)")

Swift.print("inactivity: \(self.inactivity)")

Swift.print("waiting: \(self.waiting)")

Swift.print("")

}

}

var entities: [Entity] = [Entity(seed: 0.45, previousIncomeTime: 0.0, previousProcessingEndTime: 0.0)]

for index in randomValues.indices {

let entity = entities[index]

entities.append(Entity(seed: randomValues[index], previousIncomeTime: entity.incomeTime, previousProcessingEndTime: entity.processingRange.upperBound))

}

Вычисление времени поступления и времени обслуживания:

func calculateIncomeTime(\_ seed: Double) -> Double {

if Bool.random() {

return log(2 \* seed) + 2

} else {

return -log(-2 \* seed + 2) + 2

}

}

func calculateProcessingTime(\_ seed: Double) -> Double {

return -0.5 \* log(1 - seed)

}

Случайные числа:

var randomValues = [16, 26, 98, 97, 25, 97, 14, 71, 93, 11, 88, 21, 84, 13, 81, 82, 10, 41, 69, 96, 84, 20, 30, 59, 31, 88, 17, 81, 24, 42, 67, 41, 68, 61, 40, 40, 36, 95, 53, 51, 77, 19, 19, 30, 97, 23, 52, 81, 47, 95, 86, 36, 14, 95, 73, 10, 97, 27, 79, 82, 75, 95, 39, 80, 67, 90, 43, 39, 80, 58, 85, 35, 17, 84, 75, 86, 11, 24, 59, 26, 18, 96, 70, 20, 44, 47, 70, 66, 82, 92, 74, 21, 37, 85, 94, 13, 85, 13, 65, 58, 48, 70, 92, 68, 25, 67, 86, 79, 97, 17, 54, 29, 88, 93, 91, 86, 73, 98, 51, 77, 16, 75, 80, 35, 25, 39, 67, 30, 70, 87, 20, 91, 16, 98, 46, 17, 82, 70, 83, 55, 58, 76, 35, 96, 74, 29, 55, 21].map { Double($0) / 100 }

Вычисление среднего времени обслуживания:

func calculateAverageProcessing() {

print("Average processing time: \(entities.map { $0.processingTime }.average().string)")

print("")

}

Average processing time: 0.61

Вычисление среднего времени пребывания заявки в системе:

func calculateAverageTimeInSystem() {

print("Average time in system: \(entities.map { $0.processingRange.upperBound - $0.incomeTime }.average().string)")

print("")

}

Average time in system: 0.64

Вычисление среднего количества заявок в системе:

func averageEntitiesInSystem() {

let allTime = entities.last!.processingRange.upperBound

let allProcessingTime = entities.map { $0.processingRange.upperBound - $0.processingRange.lowerBound }.reduce(0, +)

let allInactivity = entities.map { $0.inactivity }.reduce(0, +)

let allWaiting = entities.map { $0.waiting }.reduce(0, +)

let noEntities = allInactivity

let oneEntity = allProcessingTime - allWaiting

let twoEntities = allWaiting

let noEntitiesPercentage = noEntities / allTime

let oneEntityPercentage = oneEntity / allTime

let twoEntitiesPercentage = twoEntities / allTime

print("No entities: \(noEntitiesPercentage.string)%")

print("One entity: \(oneEntityPercentage.string)%")

print("Two entities: \(twoEntitiesPercentage.string)%")

let averageEntitiesInSystem = 0 \* noEntitiesPercentage + 1 \* oneEntityPercentage + 2 \* twoEntitiesPercentage

print("Average entities in system: \(averageEntitiesInSystem.string)")

print("")

var average = [0.0]

stride(from: 0.0, to: allTime, by: 0.1).forEach({ number in

average.append(Double(entities.filter({ entity in

(entity.incomeTime...entity.processingRange.upperBound).contains(number)

}).count))

})

print("Average entities in system (alternative): \(average.average())")

}

No entities: 72.92%

One entity: 25.79%

Two entities: 1.3%

Average entities in system: 28.38%

Average entities in system (alternative): 28.45%

Вычисление процента загрузки прибора:

func calculateActivityPercentage() {

let allTime = entities.last!.processingRange.upperBound

let allProcessingTime = entities.map { $0.processingRange.upperBound - $0.processingRange.lowerBound }.reduce(0, +)

let allInactivity = entities.map { $0.inactivity }.reduce(0, +)

print("Activity percentage: \((allProcessingTime / allTime).string)%")

print("Inactivity percentage: \((allInactivity / allTime).string)%")

print("Whole time: \(allTime.string)")

print("Whole activity: \(allProcessingTime.string)")

print("Whole inactivity: \(allInactivity.string)")

print("")

}

Activity percentage: 27.08%

Inactivity percentage: 72.92%

Whole time: 336.28

Whole activity: 91.06

Whole inactivity: 245.21

## 4. Вывод

Изучены методы имитационного моделирования поведения непрерывно-стохастической СМО.

## 4.1 Что такое непрерывно-стохастическая система СМО, как она описывается?

Случайный процесс называется процессом с непрерывным временем, если переход системы из состояния в состояние возможен в любой, наперёд неизвестный случайный момент времени t. То есть переходы происходят не по такту. Случайный процесс, происходящий в системе, состоит в том, что в последовательные моменты времени система оказывается в тех или иных состояниях, при этом за какой-то период времени система может и не изменить состояние.

Дискретными в такой системе являются только состояние, время же является непрерывным.

## 4.2 Опишите общую идею имитационного моделирования непрерывно-стохастической СМО?

Имитационное моделирование подразумевает использование случайных величин и законов распределения для создания модели прибора и последовательной обработки всех поступивших заявок.

## 4.3 Какие вы знаете законы распределения непрерывных случайных величин?

## 4.4 Что такое средняя загрузка обслуживающего прибора, каков ее физический смысл?