Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системный анализ и машинное моделирование (САиММод)

ОТЧЕТ

по лабораторной работе № 3-4

Тема: Аналитическое моделирование дискретно-стохастической СМО и построение её имитационной модели

Вариант 27

Выполнил

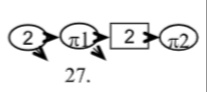
студент: гр. 651003 Шиманович А. Д.

Проверил: Мельник Н. И.

Минск 2019

**Задание 1**

Построить граф состояний P-схемы:



l = {2, 1} – количество тактов, оставшихся до поступления новой заявки

j = {0, 1, 2} – количество заявок в очереди

t1 = {0, 1} – количество заявок в канале №1

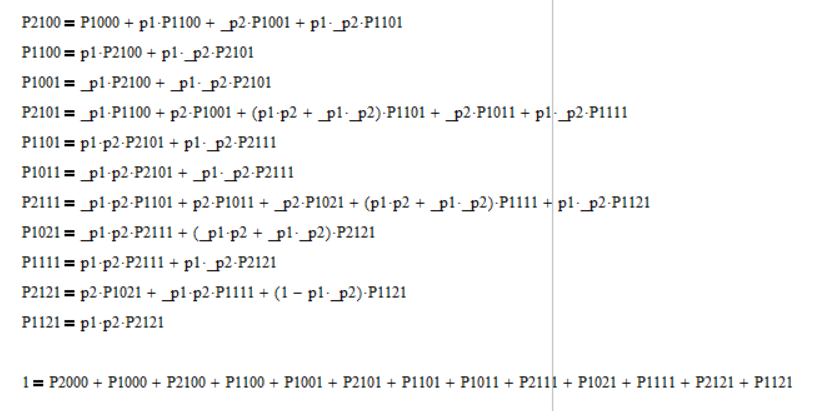
t2 = {0, 1} – количество заявок в канале №2

Общий вид кодировки состояния системы:

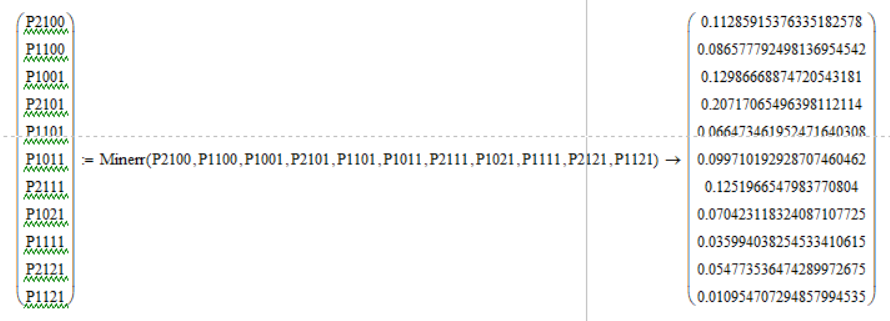
{l, t1, j, t2}



По графу построим аналитическую модель и, решив ее, определим вероятности состояний:

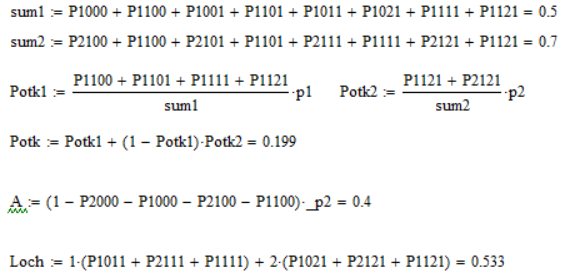


Решив систему уравнений (π1 = 0,4, π2 = 0,5), получили:



Используя данные значения, подсчитаем следующие величины:

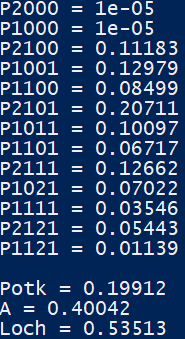
* вероятность отказа (вероятность того, что заявка, сгенерированная источником, не будет в конечном итоге обслужена системой);
* абсолютную пропускную способность **А** (среднее число заявок, обслуживаемых системой за единицу времени (такт));
* среднюю длину очереди.



**Задание 2**

Для СМО из задания 1 построить имитационную модель и исследовать ее (разработать алгоритм и написать имитирующую программу, предусматривающую сбор и статистическую обработку данных для получения оценок заданных характеристик СМО).

**Результат работы программы:**



**Листинг программы:**

**main.py:**

from Mediator import Mediator

def main():

print()

p1 = float(input('p1: '))

p2 = float(input('p2: '))

c = int(input('c: '))

print()

mediator = Mediator(p1, p2, c)

mediator.run()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Lemer.py:**

class LemerGenerator:

def \_\_init\_\_(self, m, r0, a):

self.r0 = r0

self.last\_r = r0

self.m = m

self.a = a

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

self.last\_r = (self.a \* self.last\_r) % self.m

res = self.last\_r / self.m

return res

**Task.py:**

class Task:

def \_\_init\_\_(self):

self.ticks = 0

def inc\_tick(self):

self.ticks += 1

**Source.py:**

from Task import Task

class Source:

def \_\_init\_\_(self):

self.current\_tick = 1

self.first\_time = True

def tick(self):

if self.current\_tick == 1:

self.current\_tick += 1

if not self.first\_time:

return Task()

self.first\_time = False

else:

self.current\_tick -= 1

return None

def \_\_str\_\_(self):

return str(self.current\_tick)

**TaskQueue.py:**

class TaskQueue:

def \_\_init\_\_(self, size):

self.size = size

self.tasks = []

self.sum\_of\_sizes = 0

def tick(self):

self.sum\_of\_sizes += len(self.tasks)

def enqueue(self, task):

if len(self.tasks) == self.size:

raise Exception('The queue is currently full!')

else:

self.tasks.append(task)

def dequeue(self):

task = self.tasks[0]

self.tasks = self.tasks[1:]

return task

def has\_place(self):

return len(self.tasks) < self.size

def \_\_len\_\_(self):

return len(self.tasks)

def \_\_str\_\_(self):

return str(len(self))

**Handler.py:**

import random

class Handler:

def \_\_init\_\_(self, probability, generator):

self.probability = probability

self.generator = generator

self.task = None

def tick(self):

if not self.is\_busy():

return None

# ev = next(self.generator)

ev = random.random()

if ev <= self.probability:

return None

else:

task = self.task

self.task = None

return task

def set\_task(self, task):

self.task = task

def is\_busy(self):

return self.task is not None

def \_\_str\_\_(self):

return '1' if self.is\_busy() else '0'

**Mediator.py:**

from collections import Counter

from Handler import Handler

from Lemer import LemerGenerator

from TaskQueue import TaskQueue

from Source import Source

class Mediator:

def \_\_init\_\_(self, p1 = 0.4, p2 = 0.5, iteration\_count = 100000):

self.iteration\_count = iteration\_count

self.current\_tick = 0

self.handled\_count = 0

self.refused\_count = 0

self.states = []

self.source = Source()

self.queue = TaskQueue(2)

self.handlers = [Handler(p1, LemerGenerator(209715120, 3, 7)), Handler(p2, LemerGenerator(209715120, 3, 7))]

def run(self):

self.queue.tick()

for i in range(self.iteration\_count):

self.tick()

counter = Counter(self.states)

for key in counter.keys():

counter[key] = counter[key] / self.iteration\_count

print('P{0} = {1}'.format(key, counter[key]))

print()

print('Potk = {0}'.format(2 \* self.refused\_count / self.iteration\_count))

print('A = {0}'.format(self.handled\_count / self.iteration\_count))

print('Loch = {0}'.format(self.queue.sum\_of\_sizes / self.iteration\_count))

print()

def tick(self):

self.current\_tick += 1

handler\_result = self.handlers[1].tick()

if handler\_result is not None:

self.handled\_count += 1

if (len(self.queue) > 0):

task = self.queue.dequeue()

self.handlers[1].set\_task(task)

handler\_result = self.handlers[0].tick()

if handler\_result is not None:

if not self.handlers[1].is\_busy():

self.handlers[1].set\_task(handler\_result)

else:

if self.queue.has\_place():

self.queue.enqueue(handler\_result)

else:

self.refused\_count += 1

source\_result = self.source.tick()

if source\_result is not None:

if not self.handlers[0].is\_busy():

self.handlers[0].set\_task(source\_result)

else:

self.refused\_count += 1

self.queue.tick()

state = '{0}{1}{2}{3}'.format(

str(self.source),

str(self.handlers[0]),

str(self.queue),

str(self.handlers[1])

)

self.states.append(state)

# print(state)

**Вывод:**

В ходе лабораторной работы была аналитически смоделирована дискретно-стохастическая СМО и разработана программа, имитирующая поведение данной СМО. Построенная модель позволяет статистически подсчитать характеристики СМО. Статистическое значение искомой характеристики оказывается близким к теоретически рассчитанному. Значит имитационная модель построена верно.