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

ОТЧЕТ

по лабораторным работам № 3-4

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

Вариант 27

Выполнил

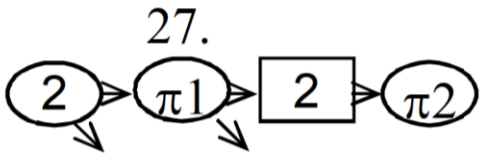
студент: гр. 751006 Калтович В. А.

Проверил: Мельник Н. И.

Минск 2020

Задание 1

Построить граф состояний Р-схемы:



p = {1, 2} – количество тактов до поступления новой заявки
t1 = {0, 1} – количество заявок в канале №1
j = {0, 1, 2} – количество заявок в очереди
t2 = {0, 1} – количество заявок в канале №2

Общий вид кодировки состояния системы:
{p, t1, j, t2}

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

$$P_{2000} = 0 \qquad P_{1000} = 0$$

$$P_{2100} = P_{1000} + P_{1100}\cdot\pi_1 + P_{1001}\cdot(1-\pi_2) + P_{1101}\cdot\pi_1\cdot(1-\pi_2)$$

$$P_{1100} = P_{2100}\cdot\pi_1 + P_{2101}\cdot\pi_1\cdot(1-\pi_2)$$

$$P_{2101} = P_{1100}\cdot(1-\pi_1) + P_{1011}\cdot(1-\pi_2) + P_{1101}\cdot[(1-\pi_1)\cdot(1-\pi_2) + \pi_1\cdot\pi_2] + P_{1001}\cdot\pi_2 + P_{1111}\cdot\pi_1\cdot(1-\pi_2)$$

$$P_{1001} = P_{2100}\cdot(1-\pi_1) + P_{2101}\cdot(1-\pi_1)\cdot(1-\pi_2)$$

$$P_{1011} = P_{2101}\cdot(1-\pi_1)\cdot\pi_2 + P_{2111}\cdot[(1-\pi_1)\cdot(1-\pi_2)]$$

$$P_{2111} = P_{1011}\cdot\pi_2 + P_{1101}\cdot(1-\pi_1)\cdot\pi_2 + P_{1111}\cdot[(1-\pi_1)\cdot(1-\pi_2) + \pi_1\cdot\pi_2] + P_{1121}\cdot[(1-\pi_2)\cdot\pi_1] + P_{1021}\cdot((1-\pi_2))$$

$$P_{1101} = P_{2111}\cdot(1-\pi_2)\cdot\pi_1 + P_{2101}\cdot\pi_1\cdot\pi_2$$

$$P_{1111} = P_{2111}\cdot\pi_1\cdot\pi_2 + P_{2121}\cdot[(1-\pi_2)\cdot\pi_1]$$

$$P_{1021} = P_{2111}\cdot(1-\pi_1)\cdot\pi_2 + P_{2121}\cdot[(1-\pi_1)\cdot(1-\pi_2) + (1-\pi_1)\cdot\pi_2]$$

$$P_{2121} = P_{1021}\cdot\pi_2 + P_{1111}\cdot[(1-\pi_1)\cdot\pi_2] + P_{1121}\cdot[(1-\pi_1)\cdot(1-\pi_2) + \pi_2\cdot\pi_1 + (1-\pi_1)\cdot\pi_2]$$

$$P_{1121} = P_{2121}\cdot\pi_1\cdot\pi_2$$

$$P_{2000} + P_{1000} + P_{2100} + P_{1100} + P_{2101} + P_{1001} + P_{1011} + P_{2111} + P_{1101} + P_{1111} + P_{1021} + P_{2121} + P_{1121} = 1$$

Решив систему уравнений (при $\pi_1 = 0{,}4$, $\pi_2 = 0{,}5$), получим:

$$\begin{pmatrix} P_{2000} \\ P_{1000} \\ P_{2100} \\ P_{1100} \\ P_{2101} \\ P_{1001} \\ P_{1011} \\ P_{2111} \\ P_{1101} \\ P_{1111} \\ P_{1021} \\ P_{2121} \\ P_{1121} \end{pmatrix} := \text{Find}\left(P_{2000}, P_{1000}, P_{2100}, P_{1100}, P_{2101}, P_{1001}, P_{1011}, P_{2111}, P_{1101}, P_{1111}, P_{1021}, P_{2121}, P_{1121}\right) \rightarrow \begin{pmatrix} 0 \\ 0 \\ 0.11285915376335182578 \\ 0.08657779249813695454 \\ 0.20717065496398112114 \\ 0.12986668874720543181 \\ 0.09971019292870746046 \\ 0.1251966547983770804 \\ 0.06647346195247164030 \\ 0.03599403825453341061 \\ 0.07042311832408710772 \\ 0.05477353647428997267 \\ 0.01095470729485799453 \end{pmatrix}$$

Используя данные значения, подсчитаем следующие величины:

- $P_{\text{отк}}$ – вероятность отказа (вероятность того, что заявка, сгенерированная источником, не будет в конечном итоге обслужена системой);
- A – абсолютная пропускная способность (среднее число заявок, обслуживаемых системой за единицу времени (такт));
- $L_{\text{оч}}$ – средняя длина очереди;
- L_c – среднее число заявок;
- Q – относительная пропускная способность (вероятность того, что заявка, сгенерированная источником, будет в конечном итоге обслужена системой);
- $W_{\text{оч}}$ – среднее время пребывания заявки в очереди;
- $W_{\text{с}}$ – среднее время пребывания заявки в системе;
- $K_{\text{кан}}$ – коэффициент загрузки канала (вероятность занятости канала).

$$A := (1 - \pi_2)\cdot(P_{2101} + P_{1001} + P_{1011} + P_{2111} + P_{1101} + P_{1111} + P_{1021} + P_{2121} + P_{1121}) = 0.4$$

$$Sum_1 := P_{1011} + P_{1111} + P_{1021} + P_{1121} + P_{1100} + P_{1001} + P_{1101} = 0.5$$

$$Sum_2 := P_{2100} + P_{1111} + P_{2101} + P_{1121} + P_{1100} + P_{2121} + P_{1101} + P_{2111} = 0.7$$

$$P_{отк1} := \frac{P_{1111} + P_{1121} + P_{1100} + P_{1101}}{Sum_1}\cdot\pi_1 = 0.16$$

$$P_{отк2} := \frac{P_{1121} + P_{2121}}{Sum_2}\cdot\pi_2 = 0.047$$

$$P_{отк} := P_{отк1} + (1 - P_{отк1})\cdot P_{отк2} = 0.199 \qquad P_{отк} := 1 - \frac{A}{\lambda} = 0.199$$

$$L_{оч} := P_{1011} + P_{2111} + P_{1111} + 2\cdot(P_{1021} + P_{2121} + P_{1121}) = 0.533$$

$$L_c := 2\cdot(P_{1011} + P_{2111} + P_{1111}) + 3\cdot(P_{1021} + P_{2121} + P_{1121}) + P_{2100} + P_{1100} + P_{1001} + P_{2101} + P_{1101} = 1.533$$

$$W_{оч} := \frac{L_{оч}}{A} = 1.332$$

$$W_c := W_{оч} + \frac{1}{1 - \pi_2} + \frac{1}{1 - \pi_1} = 4.999$$

$$Q := 1 - P_{отк} = 0.801$$

$$K_{общ} := P_{1101} + P_{1111} + P_{1121} + P_{2121} + P_{2111} + P_{2101} = 0.501$$

$$K_{кан1} := P_{1100} + K_{общ} + P_{2100} = 0.7$$

$$K_{кан2} := P_{1021} + P_{1011} + K_{общ} + P_{1001} = 0.801$$

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:

```
Enter pi1: 0.4
Enter pi2: 0.5

P2000 = 1e-05
P1000 = 1e-05
P2100 = 0.11192
P1001 = 0.12995
P2101 = 0.20676
P1100 = 0.08499
P1101 = 0.06661
P2111 = 0.1266
P1111 = 0.03717
P1011 = 0.10074
P1021 = 0.06946
P2121 = 0.05471
P1121 = 0.01107

A = 0.40081
Potk = 0.1983
Q = 0.8017

Lq = 0.53499
Lc = 1.535

Wq = 1.334697238092862
Wc = 4.83973399833749
Kkan1 = 0.69983
Kkan2 = 0.80307
```

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. Из этого следует, что имитационная модель построена верно.

Исходный код программы:

Executor.py

```
from collections import Counter

from Handler import Handler
from TaskQueue import TaskQueue
from Source import Source

class Executor:

    def __init__(self, pi1=0.4, pi2=0.5):
        self.iteration_count = 100000
        self.current_tick = 0
        self.handled_count = 0
        self.refused_count = 0
        self.states = []
        self.were_in_queue = 0
        self.were_in_sys = 0
        self.refused_by_source = 0
        self.busy0_count = 0
        self.busy1_count = 0
        self.were_in_sys_count = 0

        self.source = Source()
        self.queue = TaskQueue(2)
        self.handlers = [Handler(pi1), Handler(pi2)]

    def run(self):
        self.queue.tick()

        for i in range(self.iteration_count):
            self.tick(i)

        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('A = {0}'.format(self.handled_count / self.iteration_count))
        print('Potk = {0}'.format(2 * self.refused_count / self.iteration_count))
        print('Q = {0}'.format((self.iteration_count - 2 * self.refused_count) /
self.iteration_count))
        print()
        print('Lq = {0}'.format(self.queue.sum_of_sizes / self.iteration_count))
        print('Lc = {0}'.format(self.queue.sum_of_middle_sizes /
self.iteration_count))
        print()
        print('Wq = {0}'.format(self.were_in_queue / self.handled_count))
        print('Wc = {0}'.format((self.were_in_sys / self.were_in_sys_count)))
        print('Kkan1 = {0}'.format(self.busy0_count / self.iteration_count))
        print('Kkan2 = {0}'.format(self.busy1_count / self.iteration_count))

    def tick(self, count):
        self.current_tick += 1

        handler_result = self.handlers[1].tick()
        if handler_result is not None:
            self.handled_count += 1
            self.were_in_sys += handler_result.get_sys_ticks()
            self.were_in_sys_count += 1
```

```
            if (len(self.queue) > 0):
                task = self.queue.dequeue()
                self.were_in_queue += task.get_ticks()
                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.were_in_sys += handler_result.get_sys_ticks()
                    self.were_in_sys_count += 1
                    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_by_source += 1
                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])
        )

        if count >= self.iteration_count - 1:
            self.busy0_count = self.handlers[0].busy
            self.busy1_count = self.handlers[1].busy

        self.states.append(state)
```

Handler.py

```
import random

class Handler:

    def __init__(self, probability):
        self.probability = probability
        self.task = None
        self.busy = 0

    def tick(self):

        if not self.is_busy():
            return None

        ev = random.random()
        if ev <= self.probability:
            self.busy += 1
            self.task.inc_sys_tick()
            return None
        else:
            task = self.task
            self.task = None
```

```
            return task

    def set_task(self, task):
        self.task = task
        self.busy += 1
        self.task.inc_sys_tick()

    def is_busy(self):
        return self.task is not None

    def __str__(self):
        return '1' if self.is_busy() else '0'
```

Main.py

```
from Executor import Executor

def main():

    print()
    pi1 = float(input('Enter pi1: '))
    pi2 = float(input('Enter pi2: '))
    print()

    executor = Executor(pi1, pi2)
    executor.run()

if __name__ == "__main__":
    main()
```

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)
```

Task.py

```
class Task:

    def __init__(self):
        self.ticks = 0
        self.sys_ticks = 0

    def inc_tick(self):
        self.ticks += 1
```

```
    def get_ticks(self):
        return self.ticks

    def inc_sys_tick(self):
        self.sys_ticks += 1

    def get_sys_ticks(self):
        return self.sys_ticks
```

TaskQueue.py

```
class TaskQueue:

    def __init__(self, size):
        self.size = size
        self.tasks = []
        self.sum_of_sizes = 0
        self.sum_of_middle_sizes = 0

    def tick(self):
        for task in self.tasks:
            task.inc_tick()
            task.inc_sys_tick()
        self.sum_of_sizes += len(self.tasks)
        self.sum_of_middle_sizes += len(self.tasks) + 1

    def enqueue(self, task):
        if len(self.tasks) == self.size:
            raise Exception('The queue is 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))
```