



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 «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №4

Название: Обслуживающий аппарат

Дисциплина: Моделирование

Студент

ИУ7-75Б

(Группа)

Д.В. Сусликов

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

И.В. Рудаков

(Подпись, дата)

(И.О. Фамилия)

Москва, 2021

Задание

Необходимо промоделировать систему, состоящую из генератора, памяти, и обслуживающего аппарата. Генератор подает сообщения, распределенные по равномерному закону, они приходят в память и выбираются на обработку по закону из ЛР1 (распределение Пуассона). Количество заявок конечно и задано. Предусмотреть случай, когда обработанная заявка возвращается обратно в очередь. Необходимо определить оптимальную длину очереди, при которой не будет потерянных сообщений. Реализовать двумя способами: используя пошаговый и событийный подходы.

Теория

Равномерное распределение

Функция распределения:

$$F_X(x) \equiv P(X \leq x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}.$$

Функция плотности распределения:

$$f_X(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & x \notin [a, b] \end{cases}.$$

Пуассоновское распределение

Функция распределения:

$$F(x) = \frac{\Gamma(|k+1|, \lambda)}{|k|!}$$

Плотность распределения:

$$f(x) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Пошаговый подход

Закljučается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$. Новое состояние определяется в соответствии с их алгоритмическим описанием с учетом действия случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени. Основной недостаток: значительные затраты и опасность пропуска события при больших Δt .

Событийная модель

Состояния отдельных устройств изменяются в дискретные моменты времени. При использовании событийного принципа, состояния всех блоков системы анализируются лишь в момент возникновения какого либо события. Момент наступления следующего события, определяется минимальным значением из списка событий.

Результаты работы

The screenshot shows the 'MainWindow' application interface. It is divided into several sections: 'Параметры генератора' (Generator parameters) with inputs 'a' (1,00) and 'b' (10,00); 'Параметры процессора' (Processor parameters) with input ' λ ' (2,00); 'Свойства системы' (System properties) with inputs for 'Количество заявок' (1000), 'Вероятность повторной обработки заявки' (0), 'Метод моделирования' (Step-by-step), and 'Шаг' (0,01). The 'Результаты' (Results) section displays: 'Количество обработанных заявок' (1000), 'Количество повторно обработанных заявок' (0), 'Максимальная длина очереди' (3), and 'Время работы' (5455.04). A 'Промоделировать' (Simulate) button is at the bottom.

Параметры генератора	
a	1,00
b	10,00

Параметры процессора	
λ	2,00

Свойства системы	
Количество заявок	1000
Вероятность повторной обработки заявки	0
Метод моделирования	Пошаговый
Шаг	0,01

Результаты	
Количество обработанных заявок	1000
Количество повторно обработанных заявок	0
Максимальная длина очереди	3
Время работы	5455.04

Промоделировать

Рисунок 1 – Пошаговый при $p = 0$

The screenshot shows the 'MainWindow' application interface, identical to the previous one except for the 'Метод моделирования' (Simulation method), which is set to 'Событийный' (Event-driven). The 'Результаты' (Results) section displays: 'Количество обработанных заявок' (1000), 'Количество повторно обработанных заявок' (0), 'Максимальная длина очереди' (3), and 'Время работы' (5351.31). A 'Промоделировать' (Simulate) button is at the bottom.

Параметры генератора	
a	1,00
b	10,00

Параметры процессора	
λ	2,00

Свойства системы	
Количество заявок	1000
Вероятность повторной обработки заявки	0
Метод моделирования	Событийный
Шаг	0,01

Результаты	
Количество обработанных заявок	1000
Количество повторно обработанных заявок	0
Максимальная длина очереди	3
Время работы	5351.31

Промоделировать

Рисунок 2 – Событийный при $p = 0$

MainWindow

Параметры генератора

a 1,00 b 10,00

Параметры процессора

λ 2,00

Свойства системы

Количество заявок 1000

Вероятность повторной обработки заявки 25

Метод моделирования Пошаговый

Шаг 0,01

Результаты

Количество обработанных заявок 1000

Количество повторно обработанных заявок 228

Максимальная длина очереди 3

Время работы 4202.14

Промоделировать

Рисунок 3 – Пошаговый при $p = 25$

MainWindow

Параметры генератора

a 1,00 b 10,00

Параметры процессора

λ 2,00

Свойства системы

Количество заявок 1000

Вероятность повторной обработки заявки 25

Метод моделирования Событийный

Шаг 0,01

Результаты

Количество обработанных заявок 1000

Количество повторно обработанных заявок 216

Максимальная длина очереди 4

Время работы 4253.51

Промоделировать

Рисунок 4 – Событийный при $p = 25$

MainWindow

Параметры генератора

a: 1,00 b: 10,00

Параметры процессора

λ : 2,00

Свойства системы

Количество заявок: 1000

Вероятность повторной обработки заявки: 50

Метод моделирования: Пошаговый

Шаг: 0,01

Результаты

Количество обработанных заявок	1000
Количество повторно обработанных заявок	494
Максимальная длина очереди	7
Время работы	2733.49

Промоделировать

Рисунок 5 – Пошаговый при $p = 50$

MainWindow

Параметры генератора

a: 1,00 b: 10,00

Параметры процессора

λ : 2,00

Свойства системы

Количество заявок: 1000

Вероятность повторной обработки заявки: 50

Метод моделирования: Событийный

Шаг: 0,01

Результаты

Количество обработанных заявок	1000
Количество повторно обработанных заявок	489
Максимальная длина очереди	8
Время работы	2847.17

Промоделировать

Рисунок 6 – Событийный при $p = 50$

MainWindow

Параметры генератора

a 1,00 b 10,00

Параметры процессора

λ 2,00

Свойства системы

Количество заявок 1000

Вероятность повторной обработки заявки 99

Метод моделирования Пошаговый

Шаг 0,01

Результаты

Количество обработанных заявок	1000
Количество повторно обработанных заявок	991
Максимальная длина очереди	324
Время работы	1978.07

Промоделировать

Рисунок 7 – Пошаговый при $p = 99$

MainWindow

Параметры генератора

a 1,00 b 10,00

Параметры процессора

λ 2,00

Свойства системы

Количество заявок 1000

Вероятность повторной обработки заявки 99

Метод моделирования Событийный

Шаг 0,01

Результаты

Количество обработанных заявок	1000
Количество повторно обработанных заявок	991
Максимальная длина очереди	361
Время работы	2017.53

Промоделировать

Рисунок 8 – Событийный при $p = 99$

Вывод

Была смоделирована система, состоящая из генератора, памяти и обслуживающего аппарата.

На выходе получена максимальная длина очереди, число обрабатываемых и повторно обработанных заявок, время обработки.

Листинг

Листинг 1 – main.py

```
1 import sys
2 from PyQt5 import QtWidgets
3 from PyQt5 import uic, QtWidgets, QtGui
4 from PyQt5.QtWidgets import QApplication, QWidget, QListWidgetItem
5     , QTableWidgetItem, QMessageBox
6 import design
7 from models import event_model, step_model
8 from distributions import UniformDistribution, PoissonDistribution
9
10 class App(QtWidgets.QMainWindow, design.Ui_MainWindow):
11     def __init__(self):
12         a, b = 1, 10
13         self.generator = UniformDistribution(a, b)
14
15         lyambda = 2.0
16         self.processor = PoissonDistribution(lyambda)
17
18         self.total_tasks = 1000
19         self.repeat_percentage = 0
20         self.step = 0.01
21
22         super().__init__()
23         self.setupUi(self)
24         self.initUI()
```



```

24
25 def initUI(self):
26     self.startBtn.clicked.connect(self.startBtnPushed)
27
28
29 def startBtnPushed(self):
30     a = self.aSpinBox.value()
31     b = self.bSpinBox.value()
32     lyambda = self.lyambdaSpinBox.value()
33
34     self.generator = UniformDistribution(a, b)
35     self.processor = PoissonDistribution(lyambda)
36
37     self.total_tasks = self.tasksSpinBox.value()
38     self.repeat_percentage = self.repeatSpinBox.value()
39     self.step = self.stepSpinBox.value()
40     processed_tasks, reentered_tasks, max_queue_len, t = 0, 0,
        0, 0
41     if self.methodComboBox.currentText() == "Sobitiyniy":
42         processed_tasks, reentered_tasks, max_queue_len, t =
            event_model(self.generator, self.processor, self.
                total_tasks, self.repeat_percentage)
43     else:
44         processed_tasks, reentered_tasks, max_queue_len, t =
            step_model(self.generator, self.processor, self.
                total_tasks, self.repeat_percentage)
45
46     self.resTasksLabel.setText(str(processed_tasks))
47     self.repeatTasksLabel.setText(str(reentered_tasks))
48     self.maxLenLabel.setText(str(max_queue_len))
49     self.timeLabel.setText(str(round(t, 2)))
50
51
52 def main():
53     app = QtWidgets.QApplication(sys.argv)

```

```

54 window = App()
55 window.show()
56 app.exec_()
57
58 print('event_model:', event_model(generator, processor,
59                                     total_tasks, repeat_percentage))
60
61 print('step_model:', step_model(generator, processor,
62                                  total_tasks, repeat_percentage, step))
63
64 if __name__ == '__main__':
65     main()

```

Листинг 2 – models.py

```

1 import random
2
3
4 def step_model(generator, processor, total_tasks=0, repeat=0, step
   =0.001):
5     processed_tasks = 0
6     t_curr = step
7     t_gen_prev = t_proc = 0
8     t_gen = generator.generate()
9     t_proc = t_gen + processor.generate()
10    cur_queue_len = max_queue_len = 0
11    reentered_tasks = 0
12
13    while processed_tasks < total_tasks:
14        if t_curr > t_gen:
15            cur_queue_len += 1
16            if cur_queue_len > max_queue_len:
17                max_queue_len = cur_queue_len
18            t_gen += generator.generate()
19
20        if t_curr > t_proc:

```

```

21         if cur_queue_len > 0:
22             processed_tasks += 1
23         if random.randint(1, 100) <= repeat:
24             cur_queue_len += 1
25             reentered_tasks += 1
26         cur_queue_len -= 1
27         if cur_queue_len > 0:
28             t_proc += processor.generate()
29         else:
30             t_proc = t_gen + processor.generate()
31     t_curr += step
32
33     return processed_tasks, reentered_tasks, max_queue_len, t_curr
34
35 def event_model(generator, processor, total_tasks=0, repeat=0):
36     processed_tasks = 0
37     t_gen = generator.generate()
38     t_proc = t_gen + processor.generate()
39     cur_queue_len = max_queue_len = 0
40     reentered_tasks = 0
41     while processed_tasks < total_tasks:
42         if t_gen <= t_proc:
43             cur_queue_len += 1
44             if cur_queue_len > max_queue_len:
45                 max_queue_len = cur_queue_len
46             t_gen += generator.generate()
47
48     if t_gen >= t_proc:
49         if cur_queue_len > 0:
50             processed_tasks += 1
51             if random.randint(1, 100) <= repeat:
52                 cur_queue_len += 1
53                 reentered_tasks += 1
54             cur_queue_len -= 1
55         if cur_queue_len > 0:

```

```

56         t_proc += processor.generate()
57     else:
58         t_proc = t_gen + processor.generate()
59
60     return processed_tasks, reentered_tasks, max_queue_len, t_proc

```

Листинг 3 – distributions.py

```

1  from numpy.random import uniform, normal, poisson
2  import random
3
4  class UniformDistribution:
5      def __init__(self, a: float, b: float):
6          self.a = a
7          self.b = b
8
9      def generate(self):
10         return uniform(self.a, self.b)
11
12     class PoissonDistribution:
13         def __init__(self, lyambda):
14             self.lyambda = lyambda
15
16         def generate(self):
17             return poisson(self.lyambda)
18
19     class NormalDistribution:
20         def __init__(self, mu, sigma):
21             self.mu = mu
22             self.sigma = sigma
23
24         def generate(self):
25             return normal(self.mu, self.sigma)

```