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»

Отчет по лабораторной работе №4

Название Моделирование системы массового обслуживания

Дисциплина Моделирование

Студент Зайцева А. А.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И. В.

Москва — 2022 г.

1 Задание

Промоделировать работу системы массового обслуживания, определить минимальный размер буфера памяти, при котором не будет потерянных заявок.

Управляющую программу реализовать по двум принципам: Δt и событийному. Время появления заявок распределено по равномерному закону, время обработки заявки обслуживающим аппаратом – по закону Пуассона (вариант из лабораторной работы №1). С заданной вероятностью обработанная заявка возвращается обратно в очередь на обслуживание.

2 Теоретические сведения

2.1 Распределения

Равномерное распределение

Функция плотности распределения $f(x)$ случайной величины X, имеющей равномерное распределение на отрезке $[a, b]$ ($X \sim R(a, b)$), где $a, b \in R$, имеет следующий вид:

$$f(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & \text{иначе.} \end{cases} \tag{1}$$

Соответствующая функция распределения $F(x) = \int_{-\infty}^{x} f(t)dt$ принимает вид:

$$F(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & x \in [a, b] \\ 1, & x > b. \end{cases} \tag{2}$$

2.2 Распределение Пуассона

Дискретная случайная величина X имеет закон распределения Пуассона с параметром λ ($X \sim \Pi(\lambda)$), где $\lambda > 0$, если она принимает значения $0, 1, 2, ...$ с вероятностями:

$$P(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}, \quad k \in \{0, 1, 2, ...\} \tag{3}$$

Соответствующая функция распределения принимает вид:

$$F(x) = P(X < x) = \sum_{k=0}^{x-1} P(X = k) = e^{-\lambda}\sum_{k=0}^{x-1}\frac{\lambda^k}{k!} \tag{4}$$

2.3 Принципы реализации управляющей программы

Управляющая программа реализуется по следующим стандартным принципам.

1. Принцип Δt. Данный принцип заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$ по заданному состоянию блоков в момент времени t. При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действия случайных факторов. В результате анализа принимается решение о том, какие общесистемные события должны имитироваться на данный момент времени. Основные недостатки: значительные затраты вычислительных ресурсов при малых Δt и вероятность пропуска отдельных событий при слишком больших Δt, что исключает возможность получения правильных результатов моделирования.

2. Событийный принцип. При использовании данного принципа состояния всех блоков имитационной модели анализируются лишь в момент

появления какого либо события. Момент поступления следующего события определяется минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояния каждого из блоков системы

Помимо описанных стандартных принципов также существует комбинированный подход.

3 Результаты работы программы

В программе реализованы программные имитаторы источника информации (заявок) и обслуживающего аппарата, а также две управляющие программы, основанные на принципах Δt и событийном, соответственно. При заданных параметрах источника информации (параметры a и b равномерного распределения), параметрах обслуживающего аппарата (параметр $lambda\_value$ распределения Пуассона и вероятность повторного попадания в очередь после обработки $p\_reenter$), числе заявок $request\_count$ и шаге $delta\_t$ (для управляющей программы, основанной на принципе Δt), каждая из управляющих программ моделирует действие системы и выводит максимальную длину очереди за весь период моделирования, чтобы определить минимальный размер буфера памяти, необходимый для того, чтобы не было потерянных заявок.

Для исследования разработанная программа была выполнена при фиксированных параметрах $a = 0$, $b = 10$, $n\_tasks = 1000$, $delta\_t = 0.01$ и параметрах $lambda\_value$ и $p\_reenter$, принимающих значения из множеств $\{1, 4, 10\}$ и $\{0.1, 0.5, 0.9\}$, соотвественно. На рисунке 1 приведена таблица результатов, полученных при описанном исследовании.

lambda	вероятность повторного попадания в очередь	максимальный размер очереди
1	0.1	Событийный: 3, delta t: 3
1	0.5	Событийный: 5, delta t: 5
1	0.9	Событийный: 848, delta t: 856
4	0.1	Событийный: 8, delta t: 8
4	0.5	Событийный: 477, delta t: 480
4	0.9	Событийный: 6359, delta t: 6374
10	0.1	Событийный: 1040, delta t: 1039
10	0.5	Событийный: 2652, delta t: 2653
10	0.9	Событийный: 17075, delta t: 17089

Рисунок 1 – Таблица с результатами исследования программы

Максимальная длина очереди, полученная различными управляющими программами при одинаковых параметрах различается, так как времена генерации и обработки заявок хоть и распределены по одинаковому закону, но все еще являются случайными величинами.

Максимальная длина очереди растет по мере роста $lambda\_value$ (так как время обработки заявки растет) и $p\_reenter$ (так как все больше заявок попадают в очередь на обслуживание повторно).

4 Код программы

В листинге 1 (используемый язык – Python) приведен код программных имитаторов источника информации и обслуживающего аппарата.

Листинг 1 – Код программных имитаторов источника информации и обслуживающего аппарата

```
class RequestsGenerator:
    def __init__(self, a: float, b: float):
        self.a = a
        self.b = b

    def get_request_time(self):
        return self.a + (self.b - self.a) * random.random()


```

```
class RequestsProcessor:
    def __init__(self, lambda_value, p_reenter: float):
        self.lambda_value = lambda_value
        self.p_reenter = p_reenter
        self.queue_size = 0
        self.max_queue_size = 0
        self.processed_requests = 0

    def process_request(self):
        if self.queue_size > 0:
            self.processed_requests += 1
            self.queue_size -= 1

            if numpy_random.random_sample() <= self.p_reenter:
                self.add_request_in_queue()
                self.processed_requests -= 1

    def get_processing_time(self):
        return poisson.rvs(self.lambda_value, size=1)[0]

    def add_request_in_queue(self):
        self.queue_size += 1
        if self.queue_size > self.max_queue_size:
            self.max_queue_size = self.queue_size

    def clean_stats(self):
        self.queue_size = 0
        self.max_queue_size = 0
        self.processed_requests = 0
```

В листинге 2 приведен код управляющих программ, основанных на принципах Δt и событийном.

Листинг 2 – Код управляющих программ, основанных на принципах Δt и событийном

```
class EventBasedController:
    @staticmethod
    def simulate(generator: RequestsGenerator, processor:
   RequestsProcessor, request_count):
        gen_next_event_time = generator.get_request_time()
```

```
        proc_next_event_time = gen_next_event_time + processor.
    get_processing_time()

        while processor.processed_requests < request_count:
            if gen_next_event_time <= proc_next_event_time:
                processor.add_request_in_queue()
                gen_next_event_time += generator.get_request_time()

            else:
                processor.process_request()
                if processor.queue_size > 0:
                    proc_next_event_time += processor.get_processing_time
    ()
                else:
                    proc_next_event_time = gen_next_event_time + processor
    .get_processing_time()

        return processor.max_queue_size, round(proc_next_event_time)


class DeltaTBasedController:
    @staticmethod
    def simulate(generator: RequestsGenerator, processor:
    RequestsProcessor, request_count, delta_t):
        gen_next_event_time = generator.get_request_time()
        proc_next_event_time = gen_next_event_time
        current_time = 0
        while processor.processed_requests < request_count:
            if gen_next_event_time <= current_time:
                processor.add_request_in_queue()
                gen_next_event_time += generator.get_request_time()

            if current_time >= proc_next_event_time:
                processor.process_request()
                if processor.queue_size > 0:
                    proc_next_event_time += processor.get_processing_time
    ()
                else:
                    proc_next_event_time = gen_next_event_time + processor
    .get_processing_time()
```

```
38
39              current_time += delta_t
40
41          return processor.max_queue_size, round(current_time)
```

В листинге 3 приведен код участка программы, с помощью которого проводилось исследование.

Листинг 3 – Код участка программы, с помощью которого проводилось исследование

```
0
1  def simulate_with_params(a, b, lambda_value, reenter_prob, n_tasks,
       delta_t):
2      print()
3      print(f'Генератор: R({a}, {b}), ОА: П({lambda_value}), '
4      f'вероятность повторного попадания в очередь: {reenter_prob}, число заявок: {
       n_tasks}, delta t: {delta_t}')
5      generator = RequestsGenerator(a, b)
6      processor = RequestsProcessor(lambda_value, reenter_prob)
7
8      random.seed(0)
9      numpy_random.seed(0)
10     event_based_result = EventBasedController.simulate(generator,
       processor, n_tasks)
11
12     processor.clean_stats()
13     random.seed(0)
14     numpy_random.seed(0)
15     delta_t_based_result = DeltaTBasedController.simulate(generator,
       processor, n_tasks, delta_t)
16
17     print(f'Событийный принцип:     '
18     f'Максимальный размер очереди: {event_based_result[0]}, время окончания
       моделирования: {event_based_result[1]}')
19     print(f'Принцип delta t:        '
20     f'Максимальный размер очереди: {delta_t_based_result[0]}, время окончания
       моделирования: {delta_t_based_result[1]}')
21     return event_based_result[0], delta_t_based_result[0]
22
23
```

```
def main():
    a = 1
    b = 10
    n_tasks = 1000
    delta_t = 0.01

    lambda_values = [1, 4, 10]
    reenter_probs = [0.1, 0.5, 0.9]

    res_table = PrettyTable(field_names=['lambda', 'вероятность повторного попадания в очередь', 'максимальный размер очереди'])
    for lambda_value in lambda_values:
        for reenter_prob in reenter_probs:
            res_event, res_deltat = simulate_with_params(a, b, lambda_value, reenter_prob, n_tasks, delta_t)
            res_table.add_row([lambda_value, reenter_prob, f'Событийный: {res_event: 6d},   delta t: {res_deltat: 6d}'])
    print()
    print()
    print(res_table)
```