



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 по курсу "Моделирование"

Тема Обработка очереди

Студент Леонов В.В.

Группа ИУ7-76Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва — 2022 г.

Формальная постановка задачи

Промоделировать систему с пошаговым принципом и событийным принципом, определив минимальный размер буферной памяти, при котором не будет потерь сообщений.

Исходные данные:

- параметры генератора заявок;
- параметры обработчика заявок;
- приращение пошаговой модели;
- количество заявок;
- процент возврата заявок.

Краткие теоритические сведения

Генератор

Время прихода заявок из генератора описывается равномерным распределением.

Равномерное распределение — распределение случайной величины, принимающей значения, принадлежащей некоторому промежутку конечной длины, характеризующееся тем, что плотность вероятности в этом промежутке всюду постоянна.

Функция распределения равномерной непрерывной случайной величины имеет следующий вид:

$$F(x) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & x > b \end{cases}$$

Плотность распределения равномерной непрерывной случайной величины имеет следующий вид:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{иначе} \end{cases}$$

Обработчик

Время обработки заявок из обработчика описывается распределением Эрланга.

Распределение Эрланга — распределение, описывающее непрерывную случайную величину, принимающую неотрицательные значения и представляющую собой сумму k независимых случайных величин, распределенных по одному и тому же экспоненциальному закону с параметром θ .

Функция распределения Эрланга непрерывной случайной величины имеет следующий вид:

$$F(x) = 1 - e^{-x/\theta} \sum_{i=0}^{k-1} \frac{(x/\theta)^i}{i!}$$

Плотность распределения Эрланга непрерывной случайной величины имеет следующий вид:

$$f(x) = \frac{x^{k-1} e^{-x/\theta} \theta^k}{(k-1)!}$$

Принципы моделирования работы системы

Пошаговый принцип

Заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$ по заданному состоянию в момент t . При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени. Основным недостатком являются значительные затраты машинных ресурсов, а при недостаточном малых Δt появляется опасность пропуска события.

Событийный принцип

Состояние отдельных устройств изменяется в дискретные моменты времени, совпадающие с моментами поступления сообщения, окончания решения задачи, возникновения аварийных сигналов и т. д. При использовании событийного принципа состояния всех блоков системы анализируются лишь в момент появления какого-либо события. Момент наступления следующего события определяется минимальным значением из списка будущих событий, представляющий собой совокупность моментов ближайшего изменения состояния каждого из блоков. Момент наступления следующего события определяется минимальным значением из списка событий.

Средства реализации

Язык программирования — Python.
GUI — QT.

Листинг кода

```
1 import random
2
3 class TimeModel():
4     def __init__(self, generator, processor, step: float) -> None:
5         self.generator = generator
6         self.processor = processor
7         self.processed_tasks = 0
8         self.step = step
9
10    def start(self, total_tasks: int, repeat_rate: int) -> int:
11        t_prev = 0
12        t_current = self.step
13        t_generator = self.generator.generate()
14        t_processor = 0
15        q_current_len = 0
16        q_max_len = 0
17        q_free = True
18
19        while self.processed_tasks < total_tasks:
20            if t_current > t_generator:
21                q_current_len += 1
22                if q_current_len > q_max_len:
23                    q_max_len = q_current_len
24                t_prev = t_generator
25                t_generator += self.generator.generate()
26            if t_current > t_processor:
27                if q_current_len > 0:
28                    q_prev = q_free
29                    if q_free:
30                        q_free = False
31                    else:
32                        self.processed_tasks += 1
33                        if random.randint(0, 100) <= repeat_rate:
34                            q_current_len += 1
35                        q_current_len -= 1
36                    if q_prev:
37                        t_processor = t_prev + self.processor.generate()
38                    else:
39                        t_processor += self.processor.generate()
40                else:
41                    q_free = True
42                t_current += self.step
43        return q_max_len
```

```

1  import random
2
3  class EventModel():
4      def __init__(self, generator, processor) -> None:
5          self.generator = generator
6          self.processor = processor
7          self.processed_tasks = 0
8
9      def start(self, total_tasks: int, repeat_rate: int) -> int:
10         events = Events()
11         events.append(Event(self.generator.generate(), 'g'))
12         q_current_len = 0
13         q_max_len = 0
14         q_free = True
15         q_process = False
16
17         while self.processed_tasks < total_tasks:
18             e = events.pop(0)
19
20             if e.type == 'g':
21                 q_current_len += 1
22                 if q_current_len > q_max_len:
23                     q_max_len = q_current_len
24                 new_e = Event(e.time + self.generator.generate(), 'g')
25                 events.append(new_e)
26                 if q_free:
27                     q_process = True
28
29             elif e.type == 'p':
30                 self.processed_tasks += 1
31                 if random.randint(0, 100) <= repeat_rate:
32                     q_current_len += 1
33                     q_process = True
34
35             if q_process:
36                 if q_current_len > 0:
37                     q_current_len -= 1
38                     new_e = Event(e.time + self.processor.generate(), 'p')
39                     events.append(new_e)
40                     q_free = False
41                 else:
42                     q_free = True
43                     q_process = False
44
45         return q_max_len

```

```

1      class Event:
2          def __init__(self, time: float, type: str) -> None:
3              self.time = time
4              self.type = type
5
6
7      class Events:
8          def __init__(self) -> None:
9              self.__pool: list[Event] = []
10
11         def append(self, e: Event):
12             i = 0
13             while i < len(self.__pool) and self.__pool[i].time < e.time:
14                 i += 1
15             if 0 < i < len(self.__pool):
16                 self.__pool.insert(i - 1, e)
17             else:
18                 self.__pool.insert(i, e)
19
20         def pop(self, index):
21             return self.__pool.pop(index)

```

Демонстрация работы программы

The screenshot shows the 'Обработка очереди' (Queue Processing) application window. The interface is divided into several sections for input and output:

- Генератор (По нормальному распределению)** (Generator (By normal distribution)):
 - Левая граница (Left boundary): 0
 - Правая граница (Right boundary): 1
- Обработчик (По распределению Эрланга)** (Processor (By Erlang distribution)):
 - Альфа (Alpha): 3
 - Бета (Beta): 0.5
- Приращение пошаговой модели** (Stepwise model increment): 0.01
- Параметры моделирования** (Simulation parameters):
 - Количество заявок (Number of requests): 1000
 - Процент возврата заявок (Request return percentage): 0
- Минимально необходимый размер очереди** (Minimum required queue size):
 - пошаговой модели (stepwise model): 11026
 - событийной модели (event-driven model): 10758
- Рассчитать** (Calculate) button.

Рисунок 1 – Время обработки больше времени прихода (возврат = 0%)

The screenshot shows the 'Обработка очереди' (Queue Processing) application window with the same layout as Figure 1, but with a different return percentage:

- Генератор (По нормальному распределению)** (Generator (By normal distribution)):
 - Левая граница (Left boundary): 0
 - Правая граница (Right boundary): 1
- Обработчик (По распределению Эрланга)** (Processor (By Erlang distribution)):
 - Альфа (Alpha): 3
 - Бета (Beta): 0.5
- Приращение пошаговой модели** (Stepwise model increment): 0.01
- Параметры моделирования** (Simulation parameters):
 - Количество заявок (Number of requests): 1000
 - Процент возврата заявок (Request return percentage): 25
- Минимально необходимый размер очереди** (Minimum required queue size):
 - пошаговой модели (stepwise model): 11196
 - событийной модели (event-driven model): 11462
- Рассчитать** (Calculate) button.

Рисунок 2 – Время обработки больше времени прихода (возврат = 25%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница: 0

Правая граница: 1

Обработчик (По распределению Эрланга)

Альфа: 3

Бета: 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок: 1000

Процент возврата заявок: 50

Минимально необходимый размер очереди пошаговой модели: 11879

Минимально необходимый размер очереди событийной модели: 11458

Рассчитать

Рисунок 3 – Время обработки больше времени прихода (возврат = 50%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница: 0

Правая граница: 1

Обработчик (По распределению Эрланга)

Альфа: 3

Бета: 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок: 1000

Процент возврата заявок: 75

Минимально необходимый размер очереди пошаговой модели: 11873

Минимально необходимый размер очереди событийной модели: 11663

Рассчитать

Рисунок 4 – Время обработки больше времени прихода (возврат = 75%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница: 0

Правая граница: 1

Обработчик (По распределению Эрланга)

Альфа: 3

Бета: 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок: 1000

Процент возврата заявок: 100

Минимально необходимый размер очереди пошаговой модели

11960

Минимально необходимый размер очереди событийной модели

12128

Рассчитать

Рисунок 5 – Время обработки больше времени прихода (возврат = 100%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница: 2

Правая граница: 3

Обработчик (По распределению Эрланга)

Альфа: 1

Бета: 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок: 1000

Процент возврата заявок: 0

Минимально необходимый размер очереди пошаговой модели

10

Минимально необходимый размер очереди событийной модели

11

Рассчитать

Рисунок 6 – Время обработки меньше времени прихода (возврат = 0%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница 2

Правая граница 3

Обработчик (По распределению Эрланга)

Альфа 1

Бета 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок 1000

Процент возврата заявок 25

Минимально необходимый размер очереди пошаговой модели

80

Минимально необходимый размер очереди событийной модели

59

Рассчитать

Рисунок 7 – Время обработки меньше времени прихода (возврат = 25%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница 2

Правая граница 3

Обработчик (По распределению Эрланга)

Альфа 1

Бета 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок 1000

Процент возврата заявок 50

Минимально необходимый размер очереди пошаговой модели

275

Минимально необходимый размер очереди событийной модели

316

Рассчитать

Рисунок 8 – Время обработки меньше времени прихода (возврат = 50%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница: 2

Правая граница: 3

Обработчик (По распределению Эрланга)

Альфа: 1

Бета: 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок: 1000

Процент возврата заявок: 75

Минимально необходимый размер очереди пошаговой модели

560

Минимально необходимый размер очереди событийной модели

503

Рассчитать

Рисунок 9 – Время обработки меньше времени прихода (возврат = 75%)

Обработка очереди

Генератор (По нормальному распределению)

Левая граница: 2

Правая граница: 3

Обработчик (По распределению Эрланга)

Альфа: 1

Бета: 0.5

Приращение пошаговой модели

0.01

Параметры моделирования

Количество заявок: 1000

Процент возврата заявок: 100

Минимально необходимый размер очереди пошаговой модели

807

Минимально необходимый размер очереди событийной модели

792

Рассчитать

Рисунок 10 – Время обработки меньше времени прихода (возврат = 100%)