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 №5 по курсу "Моделирование"

Тема Центр обслуживания

Студент Леонов В.В.

Группа ИУ7-76Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва — 2022 г.

Формальная постановка задачи

Слудует выполнить моделирование процесса обслуживания клиентов в информационном центре:

- клиенты приходят в интервалы времени 10 ± 2 мин;
- клиентов обслуживают три оператора, среднее время обработки заявки которых соответственно 20 ± 5, 40 ± 10, 40 ± 20 мин;
- клиент получает отказ, если все операторы заняты;
- клиент стремится занять оператора с наивысшей производительностью;
- полученные запросы сдаются в приемный накопитель, откуда они выбираются для обработки;
- первый компьютер получает запросы от первого и второго оператора, время обработки 15 мин;
- второй компьютер получает запросы от третьего оператора, время обработки 30 мин.

Для заданного количества заявок следует определить количество и процент потерянных запросов клиентов.

Эндогенные переменные: время обработки задания на $i-$ом операторе и время решения задания на $j-$ом компьютере.

Экзогенные переменные: число обслуженных клиентов и число клиентов, получивших отказ.

Средства реализации

Язык программирования — Python.
GUI — QT.

Листинг кода

```
1  from model.base_generator import BaseGenerator
2
3  class RequestGenerator:
4      def __init__(self, generator: BaseGenerator, cnt: int) -> None:
5          self.generator = generator
6          self.cnt = cnt
7          self.receivers: RequestProcessor = []
8          self.next = 0
9
10     def generate(self) -> bool:
11         self.cnt -= 1
12         for receiver in self.receivers:
13             if receiver.check_max():
14                 return receiver
15         return None
16
17     def delay(self) -> float:
18         return self.generator.generate()
19
20 class RequestProcessor:
21     def __init__(self, generator: BaseGenerator, max_queue_size: int =
           -1) -> None:
22         self.generator = generator
23         self.queue, self.received, self.max_queue, self.processed = 0, 0,
               max_queue_size, 0
24         self.next = 0
25
26     def check_max(self) -> bool:
27         if self.max_queue == -1 or self.max_queue > self.queue:
28             self.queue += 1
29             self.received += 1
30             return True
31         return False
32
33     def process(self) -> None:
34         if self.queue > 0:
35             self.queue -= 1
36             self.processed += 1
37
38     def delay(self) -> float:
39         return self.generator.generate()
```

```
1  from model.request import RequestGenerator, RequestProcessor
2
3  class Model:
4      def __init__(self, generator: RequestGenerator, operators:
           list[RequestProcessor], computers: list[RequestProcessor]):
5          self.generator = generator
6          self.operators = operators
7          self.computers = computers
8
9      def event_mode(self):
10         lost = 0
11         generated_requests = self.generator.cnt
12         self.generator.receivers = self.operators
13         self.operators[0].receivers = [self.computers[0]]
14         self.operators[1].receivers = [self.computers[0]]
15         self.operators[2].receivers = [self.computers[1]]
16         self.generator.next = self.generator.delay()
17         self.operators[0].next = self.operators[0].delay()
18         blocks = [self.generator,
19             self.operators[0], self.operators[1], self.operators[2],
20             self.computers[0], self.computers[1]]
21
22         while self.generator.cnt >= 0:
23             current_time = self.generator.next
24             for block in blocks:
25                 if 0 < block.next < current_time:
26                     current_time = block.next
27
28             for block in blocks:
29             if current_time == block.next:
30                 if not isinstance(block, RequestProcessor):
31                     next_generator = self.generator.generate()
32                     if next_generator is not None:
33                         next_generator.next = current_time +
                               next_generator.delay()
34                     else:
35                         lost += 1
36                     self.generator.next = current_time +
                           self.generator.delay()
37                 else:
38                     block.process()
39                     if block.queue == 0:
40                         block.next = 0
41                     else:
42                         block.next = current_time + block.delay()
43         return {"lost_percentage": lost / generated_requests * 100,
44     "lost": lost}
```

```
1 from scipy.stats import uniform
2
3     class BaseGenerator:
4         def __init__(self, loc: float, scale: float = 0) -> None:
5             self.loc = loc
6             self.scale = scale
7
8         def generate(self) -> float:
9             return uniform.rvs(loc=self.loc - self.scale, scale=2 *
                  self.scale, size=1)[0]
```

Демонстрация работы программы

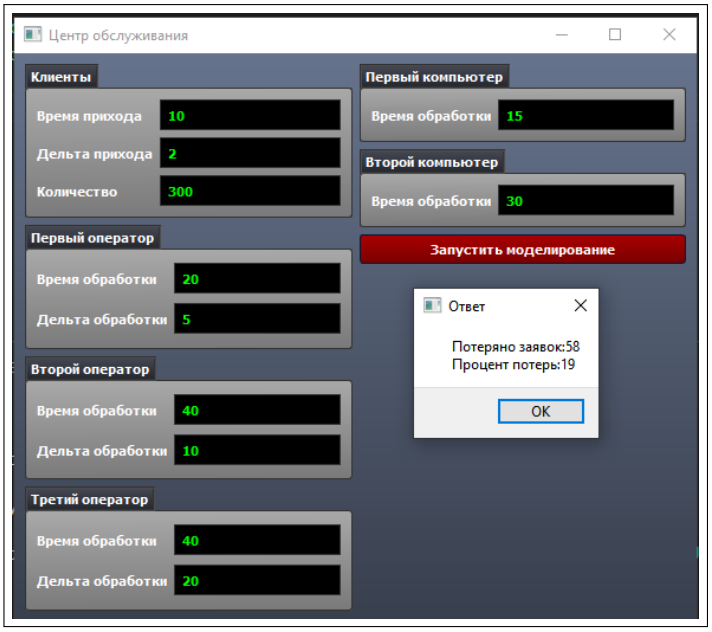

Рисунок 1 – Работа программы

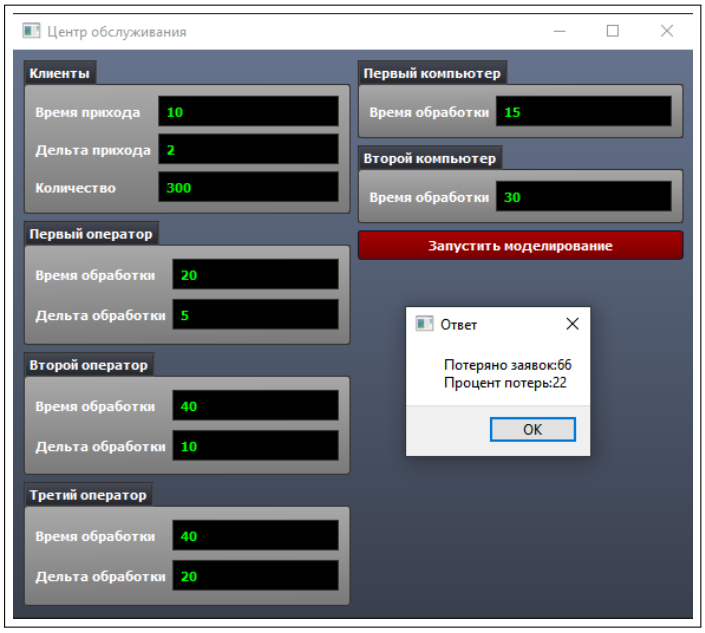

Рисунок 2 – Работа программы