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
по курсу «Моделирование»

«Моделирование процесса входа зрителей в театр»

Студент: ИУ7-73Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Преподаватель: _____ И. В. Рудаков
(подпись, дата) (И. О. Фамилия)

2022 г.

СОДЕРЖАНИЕ

1 Задание

В театр приходят зрители группами по 3 ± 2 человека через интервалы времени 5 ± 4 секунды. Вход в театр осуществляется через две двери, в каждой из которых осуществляется проверка билетов, что занимает 10 ± 5 секунд на каждого человека. С вероятностью 0.05 пришедшая группа приобрела VIP-билеты и может пройти в театр, войдя через отдельную третью дверь.

После проверки билетов и вещей каждый зритель сдает верхнюю одежду в гардероб, в котором принимают одежду 4 гардеробщика. Каждый зритель может получить номерок за 7 ± 3 секунды. Отдельного гардероба для VIP-зрителей не предусмотрено.

Известно, что зрители пришедшие группой на входе встают в одну очередь, выбирая кратчайшую. В гардеробе отдельно каждый зритель, не зависимо от группы с которой он пришел, выбирает кратчайшую очередь.

Смоделировать процесс входа 600 зрителей в театр. Определить время до начала представления, за которое необходимо начать пропуск зрителей.

2 Теоретическая часть

2.1 Схемы модели

На рисунке 2.1 представлена структурная схема модели.

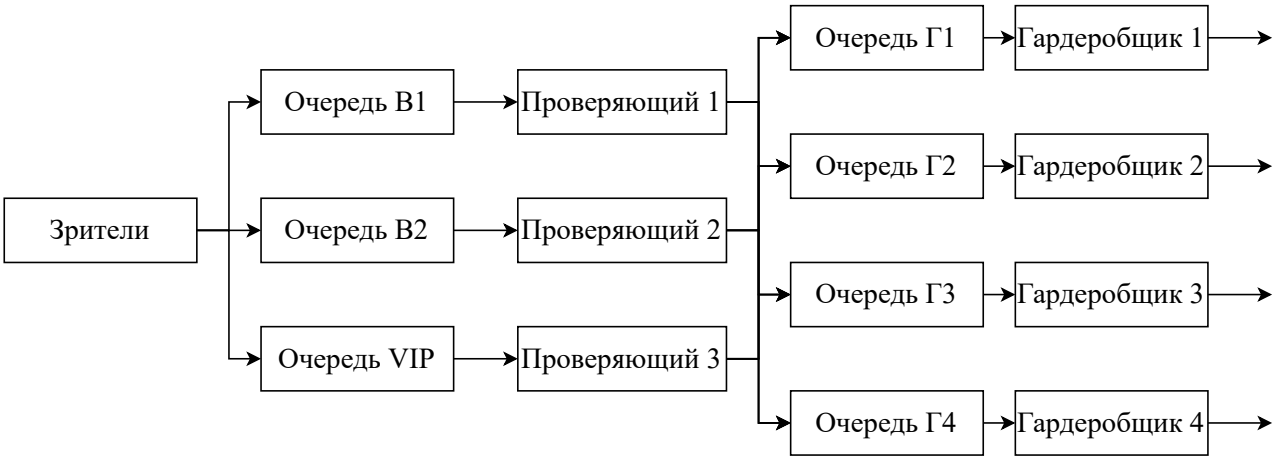

Рисунок 2.1 – Структурная схема модели

На рисунке 2.2 представлена схема модели в терминах систем массового обслуживания (СМО).

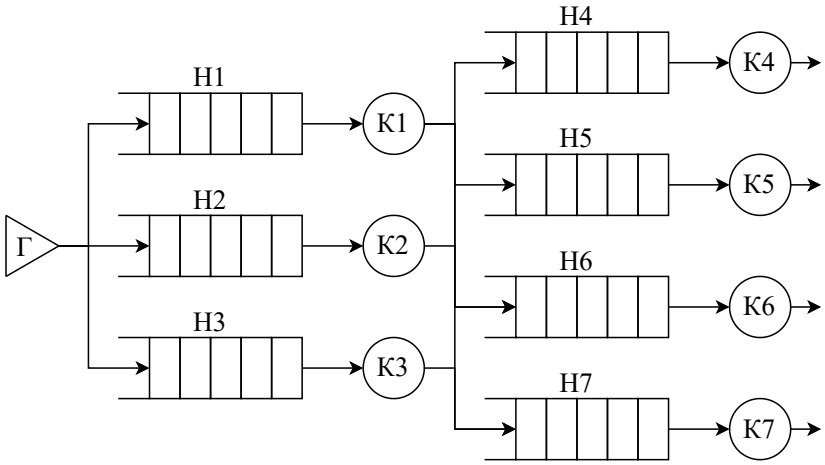

Рисунок 2.2 – Схема модели в терминах СМО

2.2 Переменные имитационной модели

Эндогенные переменные:

- время проверки билетов;
- время принятия одежды в гардероб.

Экзогенные переменные:

- время пропуска в театр всех зрителей.

3 Практическая часть

3.1 Текст программы

На листинге 3.1-3.2 представлены реализации генератора и канала обслуживания соответственно, используемые для моделирования прихода зрителей, работы проверяющих и гардеробщиков.

Листинг 3.1 — Реализация генератора

```
class Generator:

    def __init__(self, distribution: Distribution, receivers:
    ↪  list['Processor']):
        self.distribution = distribution
        self.receivers = receivers
        self.nextEvent = Event(-1, self)
        self.requestsNum = 1

    def GenerateNextEvent(self, curTime):
        self.nextEvent.Time = curTime + self.distribution.Generate()

    def TransmitRequest(self):
        if not self.receivers:
            return

        receiver = min(self.receivers, key=lambda rec:
        ↪  rec.GetQueueLength())
        return receiver.TakeRequest(self.nextEvent.time,
        ↪  self.requestsNum)

    @property
    def NextEvent(self):
        return self.nextEvent


class TheatergoersGenerator(Generator):

    def __init__(self, distribution: Distribution
                       , numDistribution:  Distribution
                       , receivers: list['ProcessorVIP']
                       , probabilityVIP: float):
        super().__init__(distribution, receivers)
        self.numDistribution = numDistribution
        self.isNextVIP = False
        self.probabilityVIP = probabilityVIP
        self.addParams = [self.isNextVIP, self.requestsNum]
```

Продолжение листинга 3.1

```
35
36      def GenerateNextEvent(self, curTime):
37          super().GenerateNextEvent(curTime)
38          self.isNextVIP = random.random() < self.probabilityVIP
39          self.requestsNum = self.numDistribution.Generate()
40
41      def TransmitRequest(self):
42          if not self.receivers:
43              return
44
45          neededRecievers = [rec for rec in self.receivers
46                                 if rec.isVIP == self.isNextVIP]
47          receiver = min(neededRecievers, key=lambda rec:
            ↪  rec.GetQueueLength())
48
49          return receiver.TakeRequest(self.nextEvent.time,
            ↪  self.requestsNum)
```

Листинг 3.2 — Реализация канала обслуживания

```
1   class Processor(Generator):
2
3       def __init__(self, generator: Generator, memory: Memory):
4           super().__init__(generator.distribution, generator.receivers)
5           self.nextEvent.eventBlock = self
6           self.memory = memory
7           self.aviable = True
8
9       def ProcessTime(self):
10          return self.GenerateNextEvent()
11
12      def SetAviable(self, state=True):
13          self.aviable = state
14
15      def IsAviable(self) -> bool:
16          return self.aviable
17
18      def GetQueueLength(self) -> int:
19          return self.memory.CurLen
20
21      def TakeRequest(self, curTime, requestsNum) -> bool:
22          if self.aviable:
23              self.SetAviable(False)
24              self.GenerateNextEvent(curTime)
25              requestsNum -= 1
26
27          if requestsNum == 0:
```

Продолжение листинга 3.2

```
            return True

        isInserted = True
        i = 0
        while isInserted and i < requestsNum:
            isInserted = self.memory.InsertRequest()
            i += 1

        return isInserted

    def EndProcess(self, curTime):
        self.TransmitRequest()

        if not self.memory.IsEmpty():
            self.memory.RemoveRequest()
            self.GenerateNextEvent(curTime)
        else:
            self.SetAviable(True)
            self.NextEvent.Time = -1


class ProcessorVIP(Processor):

    def __init__(self, generator: Generator, memory: Memory, isVIP:
    ↪ bool):
        super().__init__(generator, memory)
        self.isVIP = isVIP
```

На листинге 3.3 реализация моделирования процесс входа зрителей в театр.

Листинг 3.3 — Реализация моделирования процесса входа зрителей в театр

```
class EventModel:
    def __init__(self
                 , generator:TheatergoersGenerator
                 , checkers:  list[ProcessorVIP]
                 , cloakroomAttendant: list[Processor]
                 , theatergoersNum=1000):
        self.generator = generator
        self.checkers  = checkers
        self.attendant = cloakroomAttendant
        self.blocks = [self.generator] + self.checkers + self.attendant
        self.theatergoersNum = theatergoersNum

    def run(self):
        self.generator.GenerateNextEvent(0)
```

Продолжение листинга 3.3

```
        events = [block.NextEvent for block in self.blocks]

        theatergoersGenerated = self.generator.requestsNum
        theatergoersInTheator = 0

        curTime = 0
        while theatergoersInTheator < self.theatergoersNum:
            curTime = events[0].Time
            for event in events[1:]:
                if not (event.Time < 0) and event.Time < curTime or
                ↪  curTime < 0:
                    curTime = event.Time

            for block in self.blocks:
                if abs(block.NextEvent.Time - curTime) < EPS:
                    if not isinstance(block.NextEvent.EventBlock,
                    ↪  Processor):
                        block.TransmitRequest()

                        if theatergoersGenerated < self.theatergoersNum:
                            block.GenerateNextEvent(curTime)
                            theatergoersGenerated += block.requestsNum
                        else:
                            block.NextEvent.time = -1
                    else:
                        block.EndProcess(curTime)
                        if not isinstance(block, ProcessorVIP):
                            theatergoersInTheator += 1

        return curTime
```

3.2 Полученный результат

На рисунке 3.1 приведен примеры работы программы на значениях, данных в условии.

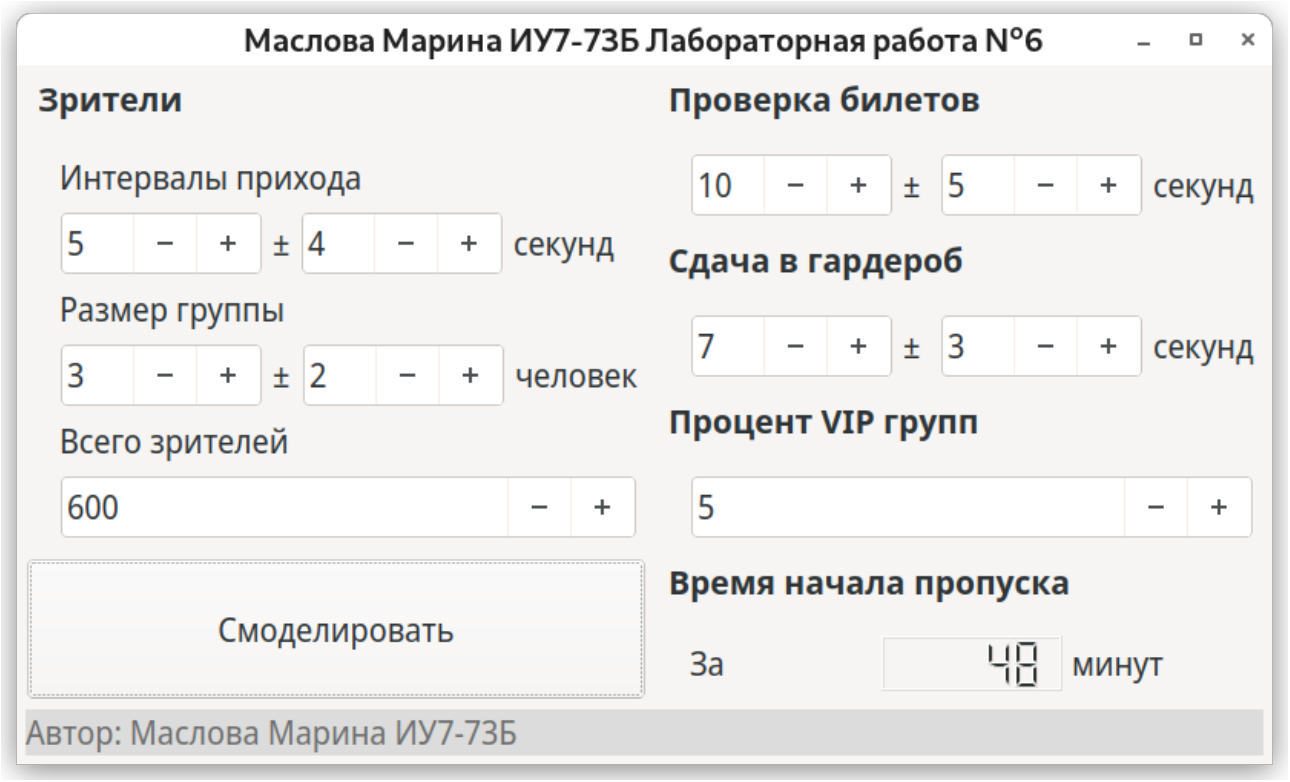

Рисунок 3.1 – Пример работы программы