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 по дисциплине «Моделирование»

Тема Алгоритмы продвижения модельного времени

Студент Слепокурова М.Ф.

Группа ИУ7-76Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва — 2023 г.

Постановка задачи

Промоделировать систему, состоящую из источника информации (ИИ), буферной памяти (БП) и обслуживающего аппарата (ОА), используя принцип Δt и событийный алгоритм. Источник информации генерирует заявки, время появления которых распределено по равномерному закону, а обслуживающий аппарат обрабатывает каждую из них за время, распределенное по закону Эрланга. Определить минимальный размер буферной памяти, при котором не будет потерь заявок. Учесть возможность задания вероятности повторного попадания заявок из обслуживающего аппарата в очередь.

Теория

Принцип Δt

Данный принцип заключается в последовательном анализе состояний всех блоков 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 действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на данный момент времени.

Основной недостаток принципа — значительные затраты вычислительных ресурсов, а при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, исключающая возможность получения правильных результатов при моделировании.

К достоинствам метода можно отнести равномерную протяжку модельного времени в виду фиксированного временного интервала Δt.

Событийный принцип

Характерное свойство систем обработки информации заключается в том, что состояния отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему, окончания реализации процесса, возникновения прерываний и аварийных сигналов и т.д. Поэтому моделирование и продвижение времени в системе удобно проводить, используя событийный принцип, при котором состояние всех блоков имитационной модели анализируется 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.

Равномерное распределение

Равномерное распределение описывает случайную величину, принимающую значения, принадлежащие некоторому промежутку конечной длины, при этом 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:

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

В качестве параметров по умолчанию для равномерного распределения использовались значения $a = 1$ и $b = 10$.

Распределение Эрланга

Распределение Эрланга описывает непрерывную случайную величину, принимающую неотрицательные значения и представляющую собой сумму n независимых случайных величин, распределенных по одному и тому же экспоненциальному закону с параметром λ.

Функция распределения Эрланга непрерывной случайной величины имеет следующий вид:

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

Плотность распределения Эрланга непрерывной случайной величины имеет следующий вид:

$$f(x) = \frac{x^{n-1} e^{-x\lambda} \lambda^n}{(n-1)!}$$

В качестве параметров по умолчанию для распределения Эрланга использовались значения $n = 9$ и $\lambda = 0.5$.

Средства реализации

Для реализации приложения был выбран язык программирования Python.

Листинг кода

```
class RequestGenerator:
  def __init__(self, generator):
    self._generator = generator
    self._receivers = set()

  def addReceiver(self, receiver):
    self._receivers.add(receiver)

  def removeReceiver(self, receiver):
    if (receiver in self._receivers):
      self._receivers.remove(receiver)

  def nextTimePeriod(self):
    return self._generator.generateTime()

  def emitRequest(self):
    for receiver in self._receivers:
      receiver.receiveRequest()

class RequestProcessor():
  def __init__(self, generator, reenterProbability=0):
    self._generator = generator
    self._currQueueSize = 0
    self._maxQueueSize = 0
    self._processedRequests = 0
    self._reenterProbability = reenterProbability
    self._reenteredRequests = 0

  def processedRequests(self):
    return self._processedRequests

  def maxQueueSize(self):
    return self._maxQueueSize

  def currQueueSize(self):
    return self._currQueueSize

  def reenteredRequests(self):
    return self._reenteredRequests

  def process(self):
    if self._currQueueSize > 0:
      self._processedRequests += 1
      self._currQueueSize -= 1
      if nr.random_sample() < self._reenterProbability:
        self._reenteredRequests += 1
```

```
        self.receiveRequest()

  def receiveRequest(self):
    self._currQueueSize += 1
    if self._currQueueSize > self._maxQueueSize:
      self._maxQueueSize += 1

  def nextTimePeriod(self):
    return self._generator.generateTime()

class Modeller:
  def __init__(self, generator, processor):
    self._generator = generator
    self._processor = processor
    self._generator.addReceiver(self._processor)

  def eventBasedModelling(self, requestCount):
    generator = self._generator
    processor = self._processor

    genPeriod = generator.nextTimePeriod()
    procPeriod = genPeriod + processor.nextTimePeriod()
    while processor.processedRequests() < requestCount:
      if genPeriod <= procPeriod:
        generator.emitRequest()
        genPeriod += generator.nextTimePeriod()
      else:
        processor.process()
        if processor.currQueueSize() > 0:
          procPeriod += processor.nextTimePeriod()
        else:
          procPeriod = genPeriod + processor.nextTimePeriod()

    return { "processedRequests": processor.processedRequests(),
             "reenteredRequests": processor.reenteredRequests(),
             "maxQueueSize": processor.maxQueueSize() }

  def timeBasedModelling(self, requestCount, dt=1):
    generator = self._generator
    processor = self._processor

    genPeriod = generator.nextTimePeriod()
    procPeriod = genPeriod + processor.nextTimePeriod()
    currTime = 0
    while processor.processedRequests() < requestCount:
      if genPeriod <= currTime:
        generator.emitRequest()
        genPeriod += generator.nextTimePeriod()
      if procPeriod <= currTime:
```

```
        processor.process()
        if processor.currQueueSize() > 0:
          procPeriod += processor.nextTimePeriod()
        else:
          procPeriod = genPeriod + processor.nextTimePeriod()
      currTime += dt

    return { "processedRequests": processor.processedRequests(),
             "reenteredRequests": processor.reenteredRequests(),
             "maxQueueSize": processor.maxQueueSize() }

generator = RequestGenerator(UniformGenerator())
processor = RequestProcessor(ErlangGenerator(), REENTER_PROBABILITY)
model = Modeller(generator, processor)
resultTimeBased = model.timeBasedModelling(REQUEST_COUNT, DELTA_T)
print("Time algorithm:")
print("-" * 26)
print("Processed requests: ", resultTimeBased["processedRequests"])
print("Reenter probability: ", REENTER_PROBABILITY)
print("Reentered requests: ", resultTimeBased["reenteredRequests"])
print("Buffer memory size: ", resultTimeBased["maxQueueSize"])
print("\n")

generator = RequestGenerator(UniformGenerator())
processor = RequestProcessor(ErlangGenerator(), REENTER_PROBABILITY)
model = Modeller(generator, processor)
resultEventBased = model.eventBasedModelling(REQUEST_COUNT)
print("Event algorithm:")
print("-" * 26)
print("Processed requests: ", resultEventBased["processedRequests"])
print("Reenter probability: ", REENTER_PROBABILITY)
print("Reentered requests: ", resultEventBased["reenteredRequests"])
print("Buffer memory size: ", resultEventBased["maxQueueSize"])
```

Демонстрация работы программы

На рисунке 1 изображен пример работы программы для 10000 заявок с вероятностью повторного попадания заявки в очередь равной 0.

```
Time algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0
Reentered requests:  0
Buffer memory size:  8


Event algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0
Reentered requests:  0
Buffer memory size:  8
```

Рисунок 1 – Пример работы программы — 1

На рисунке 2 изображен пример работы программы для 10000 заявок с вероятностью повторного попадания заявки в очередь равной 0.1.

```
Time algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.1
Reentered requests:  1019
Buffer memory size:  19


Event algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.1
Reentered requests:  1046
Buffer memory size:  18
```

Рисунок 2 – Пример работы программы — 2

На рисунке 3 изображен пример работы программы для 10000 заявок с вероятностью повторного попадания заявки в очередь равной 0.3.

```
Time algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.3
Reentered requests:  3040
Buffer memory size:  1262


Event algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.3
Reentered requests:  3073
Buffer memory size:  1220
```

Рисунок 3 – Пример работы программы — 3

На рисунке 4 изображен пример работы программы для 10000 заявок с вероятностью повторного попадания заявки в очередь равной 0.5.

```
Time algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.5
Reentered requests:  4978
Buffer memory size:  3191


Event algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.5
Reentered requests:  4949
Buffer memory size:  3178
```

Рисунок 4 – Пример работы программы — 4

На рисунке 5 изображен пример работы программы для 10000 заявок с вероятностью повторного попадания заявки в очередь равной 0.7.

```
Time algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.7
Reentered requests:  7042
Buffer memory size:  5175


Event algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  0.7
Reentered requests:  6909
Buffer memory size:  5123
```

Рисунок 5 – Пример работы программы — 5

На рисунке 6 изображен пример работы программы для 10000 заявок с вероятностью повторного попадания заявки в очередь равной 1.

```
Time algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  1
Reentered requests:  10000
Buffer memory size:  8183


Event algorithm:
-------------------------
Processed requests:  10000
Reenter probability:  1
Reentered requests:  10000
Buffer memory size:  8190
```

Рисунок 6 – Пример работы программы — 6