Белорусский государственный университет информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Отчет по лабораторным работам № 5, 6

по дисциплине «Системный анализ и машинное моделирование»

Вариант 25

Выполнил:

студент группы 451006

Снитовец М. В.

Проверил:

Мельник Н. И.

**1. Построение и исследование аналитической модели непрерывно-стохастической системы массового обслуживания**

В двухканальной СМО вида M/M/n интенсивность обслуживания первого канала 1=5, второго 2. Определить значение 2 при котором будет обслуживаться не менее 95% поступающих заявок, если=9,5. Все потоки простейшие.

|  |  |
| --- | --- |
| c1 | = {0, 1} – количество заявок в канале 1; |
| c2 | = {0, 1} – количество заявок в канале 2; |

Общий вид кодировки состояния системы: **{c1, c2}** по графу построить аналитическую модель и, решив ее, определить вероятности состояний:

*(*

*(*

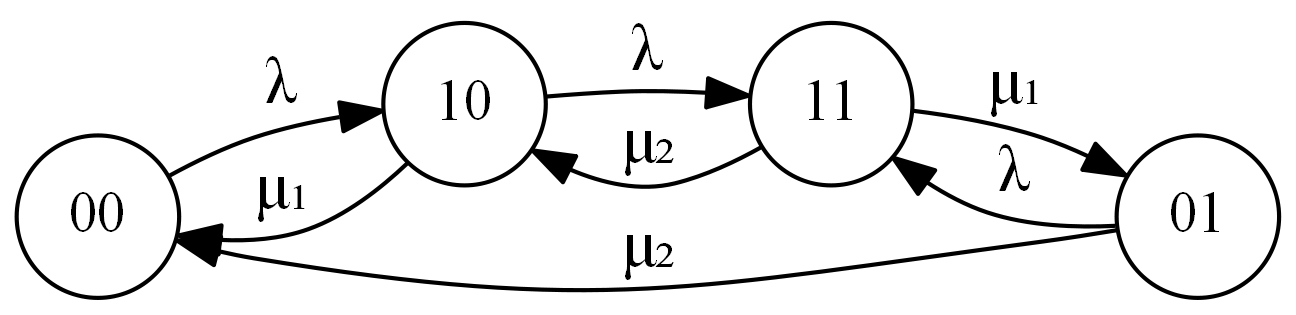
*(*

Учитывая условие, относительная пропускная способность системы

Решив систему уравнений (при , ), получим:

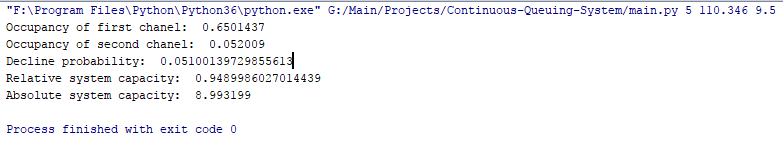
Теоретические значения показателей:

|  |  |
| --- | --- |
|  |  |

****

**2. Построение и исследование имитационной модели непрерывно-стохастической системы массового обслуживания**

* для заданной СМО простроить имитационную модель и исследовать ее:

Результаты работы программы (имитация cто тысячи тактов):

**Вывод**:

В ходе лабораторной работы была аналитически смоделирована непрерывно- стохастическая СМО и разработана программа, имитирующая ее поведение. Построенная модель позволяет статистически подсчитать необходимые характеристики СМО. Полученные статистические значения искомых характеристик близки к теоретически рассчитанным: отклонения при моделировании ста тысяч тактов составляют не более нескольких сотых.

Исходный код программы:

**from** arguments **import** parse\_arguments  
**from** channel **import** Channel  
**from** generator **import** Generator  
  
mu1 = 5  
mu2 = 110.379  
lmbd = 9.5  
normalizing\_factor = 0  
accuracy = 100  
  
  
**def** main():  
 arguments = parse\_arguments()  
 normailze\_inputs(arguments)  
  
 first\_chanel = Channel(mu1)  
 second\_chanel = Channel(mu2)  
 generator = Generator(lmbd)  
 ticks\_number = 100000 \* accuracy  
 declined\_claims = 0  
 generated\_claims = 0  
 processed\_claims = 0  
 **for** i **in** range(0, ticks\_number):  
 **if** generator.is\_generated():  
 generator.start\_generate()  
 generated\_claims += 1  
 **if** first\_chanel.is\_processed():  
 first\_chanel.add()  
 processed\_claims += 1  
 **elif** second\_chanel.is\_processed():  
 second\_chanel.add()  
 processed\_claims += 1  
 **else**:  
 declined\_claims += 1  
 first\_chanel.tick()  
 second\_chanel.tick()  
 generator.tick()  
  
 print(**'Occupancy of first chanel: '**, first\_chanel.work\_time / ticks\_number)  
 print(**'Occupancy of second chanel: '**, second\_chanel.work\_time / ticks\_number)  
 print(**'Decline probability: '**, declined\_claims / generated\_claims)  
 print(**'Relative system capacity: '**, processed\_claims / generated\_claims)  
 print(**'Absolute system capacity: '**, processed\_claims  
 \* normalizing\_factor / ticks\_number)  
  
  
**def** normailze\_inputs(arguments):  
 **global** mu1  
 **global** mu2  
 **global** lmbd  
 **global** normalizing\_factor  
 mu1 = arguments.mu1  
 mu2 = arguments.mu2  
 lmbd = arguments.lmbd  
 normalizing\_factor = max(mu1, mu2, lmbd) \* accuracy  
 mu1 = mu1 / normalizing\_factor  
 mu2 = mu2 / normalizing\_factor  
 lmbd = lmbd / normalizing\_factor  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

**def** exponential\_number(number):  
 **return** (-1 / number) \* log(uniform(0, 1))

**class** Generator:  
 **def** \_\_init\_\_(self, intensity):  
 self.\_intensity = intensity  
 self.\_ticks\_for\_generate = 0  
  
 **def** start\_generate(self):  
 self.\_ticks\_for\_generate = exponential\_number(self.\_intensity)  
  
 **def** is\_generated(self):  
 **return** self.\_ticks\_for\_generate <= 0  
  
 **def** tick(self):  
 **if** self.\_ticks\_for\_generate > 0:  
 self.\_ticks\_for\_generate -= 1

**class** Channel:  
 **def** \_\_init\_\_(self, intensity):  
 self.\_intensity = intensity  
 self.\_ticks\_for\_process = 0  
 self.work\_time = 0  
 self.processed\_claims = 0  
 self.is\_first\_claim = **True  
  
 def** add(self):  
 self.\_ticks\_for\_process = exponential\_number(self.\_intensity)  
 **if** self.is\_first\_claim:  
 self.is\_first\_claim = **False  
 else**:  
 self.processed\_claims += 1  
  
 **def** is\_processed(self):  
 **return** self.\_ticks\_for\_process <= 0  
  
 **def** tick(self):  
 **if** self.\_ticks\_for\_process > 0:  
 self.\_ticks\_for\_process -= 1  
 self.work\_time += 1