|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

**ОТЧЕТ ПО 3-МУ МОДУЛЮ ПРОЕКТНО- ТЕХНОЛОГИЧЕСКОЙ ПРАКТИКИ**

по дисциплине: «Проектно-технологическая практика»

|  |  |  |
| --- | --- | --- |
| Студент |  | Горохова Александра |
| Группа |  | РК6-36Б |
| Тип задания |  | Проектно-технологическая практика |
| Тема лабораторной работы |  |  |

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Горохова А.С.**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Берчун Ю.В.**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2021 г*.

**Оглавление**

[**Задание** 3](#_Toc92756256)

[**Описание выходных данных** 3](#_Toc92756257)

[**Алгоритм** 3](#_Toc92756258)

[**Приложение 1. Листинг программы** 5](#_Toc92756259)

# **Задание**

Требуется разработать программу, реализующую дискретно-событийное моделирование системы, рассмотренной в задании 2 домашнего задания №4. Обратите внимание, что все интервалы времени подчиняются законам распределений, носящим непрерывный характер. Поэтому категорически неверными является выбор целочисленных типов данных для моментов и интервалов времени, и тем более инкремент модельного времени с единичным шагом. Нужно реализовать именно переход от события к событию, как это сделано в GPSS и других проблемно-ориентированных системах. Для упрощения можно ограничиться использованием единственного потока случайных чисел для генерации всех необходимых случайных величин. Результатом работы программы должен быть лог-файл, содержащий записи типа: «В момент времени 12.345 транзакт с идентификатором 1 вошёл в модель», «В момент времени 123.456 транзакт с идентификатором 123 встал в очередь 1», «В момент времени 234.567 транзакт с идентификатором 234 занял устройство 2», «В момент времени 345.678 транзакт с идентификатором 345 освободил устройство 1», «В момент времени 456.789 транзакт с идентификатором 456 вышел из модели».

# **Описание выходных данных**

Log-файл приложен.

# **Алгоритм**

Данная программа основана на принципе работы проблемно-ориентированных систем. В таких системах реализуется переход от события к событию, в этом случае возможны следующие события:

1. Приход новой заявки в систему – в любое время на промежутке от прихода предыдущей заявки до заранее обозначенного ограничения
2. Начало обслуживания заявки на одном из устройств. Длительность обслуживания также располагается в границах заранее заданного интервала.
3. Постановка заявки в очередь – в случае, если оба устройства заняты. Стоит заметить, что невозможен переход между очередями, даже если другое устройство закончило заявку раньше
4. Завершения обслуживания заявки и выход заявки из системы

Каждая заявка проходит цикл 1-2-4 либо 1-2-3-4. Работа системы продолжается до тех пор, пока не будет достигнуто заданное пользователем время. В нашем случае это 1 час. Так же стоит отметить, что при условии непрерывности времени системы, будем считать события, произошедшие в один момент времени, невозможными.

Инициализация системы начинается с задания её параметров:

**time –** счетчик времени системы, увеличивается при переходе от события к событию;

**tranzact\_num** – номер заявки в системе, инкрементируется с каждой новой заявкой;

**max\_sec –** максимально допустимое время интервала между заявкой и максимально допустимое время обслуживания заявки на устройстве;

**station\_time –** массив из двух значений, минимально допустимое время обслуживания заявки на каждом из устройств.

Устройства и их очередь реализованы как двумерный массив **que**, где первый элемент каждого из подмассивов отвечает за обслуживаемую на данный момент заявку, а следующие формируют очередь. В этом массиве содержатся номера заявок.

Массив **all\_time** формируется из трёх элементов – времени поступления новой заявки, времени обслуживания заявок на первом и втором устройстве. Служит для определения очередности событий.

Базовая оболочка системы строится на бесконечном цикле, выход из которого происходит при достижении максимального времени. На каждой итерации ищется событие, время для которого ближе всего, с учетом этого обновляется временной массив и запускается функция обновления состояния **update\_stat**

**update\_stat** имеет два варианта действий в зависимости от минимального найденного времени:

1. Минимальное время – время прихода заявки в систему, тогда она отправляется в одну из очередей устройств (или на само устройство, это решает функция **que\_upd**, которая будет рассмотрена ниже)
2. Минимальное время – время обслуживания заявки на одном из устройств.

Предположим, что произошёл первый вариант, и минимальным является время прихода заявки. Тогда, в зависимости от того, какая очередь короче (при их отсутствии или равенстве заявка отправится на первое устройство), вызывается функция **que\_upd**. В ней также имеется два варианта:

1. Если очереди на устройстве нет – заявка отправляется на обслуживание
2. Если очередь есть – заявка отправляется в её конец

Теперь рассмотрим вариант, когда ближайшим событием является окончание обслуживания заявки на одном из устройств. Функция **tranz\_upd** записывает в файл время окончания обслуживания, выхода заявки из системы и номера заявки, а так же обновляет временной массив и очередь устройства.

Таким образом, программа реализует переход от события к событию, постепенно увеличивая время до максимально возможного.

# **Приложение 1. Листинг программы**

import random

rnd=random.randint(0,100)

random.seed(rnd) # воспроизводимость результатов

time = 0 # глобальное время модели

tranzact\_num = 1 # номер транзакта

max\_sec = 24 # максимально допустимое время для одного события

station\_time = [9, 8]

all\_time = [random.uniform(0, max\_sec), 0, 0] # время текущего транзакта и двух устройств

que = [[], []] # обслуживающие устройства + очередь

log = open('log.txt', 'w') # открытие файла для записи

def min\_time(): # поиск минимального времени

if len(que[0]) == 0 and len(que[1]) == 0: # при нулевых очередях - время поступления заявки

return [all\_time[0], 0]

elif len(que[0]) == 0: # при отсутствии первой очереди - время поступления либо время второй очереди

return [min(all\_time[0], all\_time[2]), all\_time.index(min(all\_time[0], all\_time[2]))]

elif len(que[1]) == 0: # при отсутствии второй очереди - время поступления либо время первой очереди

return [min(all\_time[0], all\_time[1]), all\_time.index(min(all\_time[0], all\_time[1]))]

else: # если есть обе очереди - минимальное из всех

local = 0

local\_min = min(all\_time)

for i in all\_time:

if local\_min == i:

local = all\_time.index(i)

return [local\_min, local]

def update\_stat(i): # обновление состояний

global tranzact\_num

if i == 0: # минимальное время - время новой заявки

log.write("В момент времени " + str(time) + " транзакт с идентификатором " + str(tranzact\_num) + " вошел в модель\n")

all\_time[0] = random.uniform(0, max\_sec) # обновление времени заявки

if len(que[0]) <= len(que[1]): # условие на очередь

que\_upd(1)

else:

que\_upd(2)

tranzact\_num += 1

else: # минимальное время - время выхода заявки из одного из устройств

tranz\_upd(i)

def que\_upd(i): # обновление очереди

if len(que[i-1]) == 0: # заявка сразу идет на обслуживание

log.write("В момент времени " + str(time) + " транзакт с идентификатором " +str(tranzact\_num) + " занял устройство " + str(i) + "\n")

all\_time[i] = random.uniform(station\_time[i-1], max\_sec) # обновление времени обслуживания

else:

log.write("В момент времени " + str(time) + " транзакт с идентификатором " +str(tranzact\_num) + " встал в очередь " + str(i) + "\n")

que[i-1].append(tranzact\_num) # добавляем в очередь номер заявки

def tranz\_upd(i): # обслуживание заявки на устройстве

log.write("В момент времени " + str(time) + " транзакт с идентификатором " +str(que[i-1][0]) + " освободил устройство " + str(i) + "\n")

log.write("В момент времени " + str(time) + " транзакт с идентификатором " +str(que[i-1][0]) + " вышел из модели\n")

que[i-1].pop(0) # удаление заявки из системы

if len(que[i-1]) > 0: # если есть очередь - обновляем глобальное время устройства

log.write("В момент времени " + str(time) + " транзакт с идентификатором " +str(que[i-1][0]) + " занял устройство " + str(i) + "\n")

all\_time[i] = random.uniform(station\_time[i-1], max\_sec)

else: # если нет - делаем его нулевым

all\_time[i] = 0

while True:

minimal = min\_time()

for i in range(3): # обновление времени с учетом найденного минимального.

if all\_time[i] != 0:

all\_time[i] -= minimal[0]

time += minimal[0]

if time > 3600:

log.write("Время симуляции, равное 3600 секунд, закончено")

log.close()

break

update\_stat(minimal[1])