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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение эвм и информационные технологии»

ОТЧЕТ

По лабораторной работе №4

«Моделирование простейшего прибора обслуживания»

По курсу «Моделирование»

Студент	Т. А. Казаева
Группа	ИУ7-76Б
Преподаватель	И. В. Рудаков

2022 г.

1. Задание

Смоделировать систему, состоящую из генератора, памяти, и обслуживающего аппарата.

Генератор подает сообщения, распределенные по равномерному закону, они приходят в память и выбираются на обработку по нормальному (Гауссовому) закону.

Количество заявок конечно и задано. Предусмотреть случай, когда обработанная заявка возвращается обратно в очередь. Необходимо определить оптимальную длину очереди, при которой не будет потерянных сообщений. Реализовать двумя способами: используя пошаговый и событийный подходы.

2. Математическая формализация

2.1 Законы распределения

2.1.1 Непрерывное равномерное распределение

Говорят, что случайная величина X имеет непрерывное равномерное распределение на отрезке $[a, b]$, если её функция плотности имеет вид (2.1):

$$f_X(x) = \begin{cases} \dfrac{1}{b-a}, & x \in [a, b] \\ 0, & x \notin [a, b] \end{cases}. \tag{2.1}$$

Обозначается: $X \sim U[a, b]$.
Функция распределения равномерной случайной величины $X \sim U[a, b]$ (2.2):

$$F_X(x) \equiv \mathbb{P}(X \leqslant x) = \begin{cases} 0, & x < a \\ \dfrac{x-a}{b-a}, & a \leqslant x < b \\ 1, & x \geqslant b \end{cases} \cdot \tag{2.2}$$

2.1.2 Нормальное распределение

Говорят, что случайная величина X имеет нормальное распределение с параметрами μ и σ^2 $(\sigma^2 > 0)$, если её функция плотности имеет вид (2.3):

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\dfrac{(x-\mu)^2}{2\sigma^2}}, x \in \mathbb{R} \tag{2.3}$$

Обозначается: $X \sim N(\mu, \sigma^2)$.

Функция распределения нормальной случайной величины $X \sim N(\mu, \sigma^2)$:

$$\frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right], \tag{2.4}$$

где

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{2.5}$$

2.2 Управляющая программа индукционной модели

2.2.1 Пошаговый метод

Заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$. Новое состояние определяется в соответствии с их алгоритмическим описанием с учетом действия случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени.

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

2.2.2 Событийный метод

Состояния отдельных устройств изменяются в дискретные моменты времени. При использовании событийного принципа, состояния всех блоков системы анализируются лишь в момент возникновения какого либо события. Момент наступления следующего события определяется минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояния каждого из блоков системы.

3. Результат

3.1 Программный интерфейс

Пользователю предоставляется следующие способы ввода занчений в программу:

- с помощью аргументов командной строки (рис. 3.1), документация к которым может быть получена с помощью специальной команды (рис. 3.3)
- с помощью интерфейса непосредственно – пиглашения на ввод поступают последовательно после нажатия клавиши «Enter» с соответсвующими комментариями (рис. 3.2).

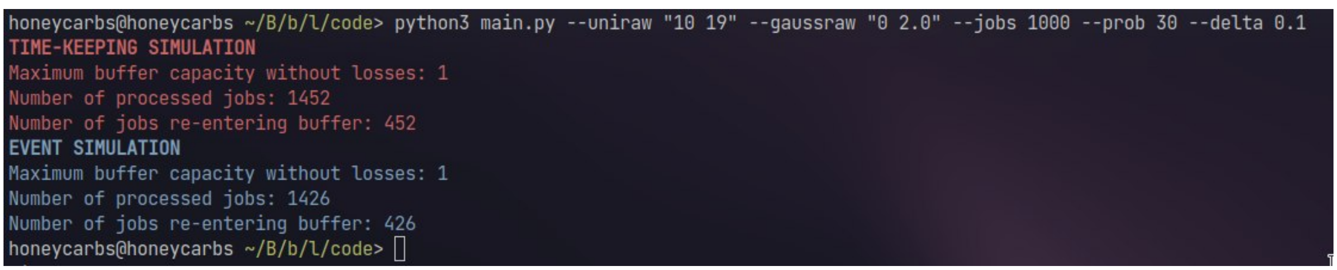


Рис. 3.1: Ввод данных с помощью аргументов командной строки

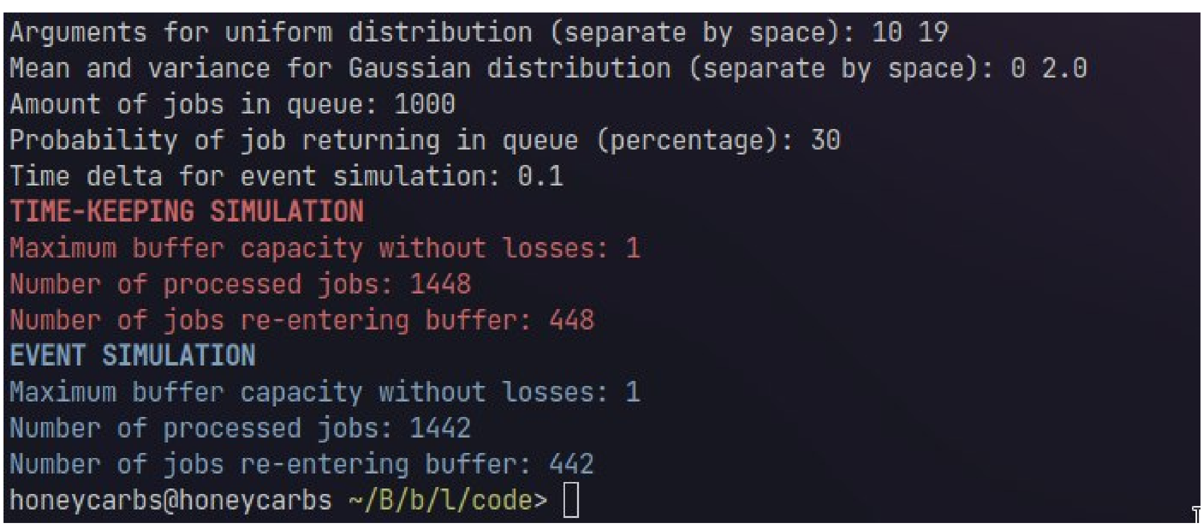
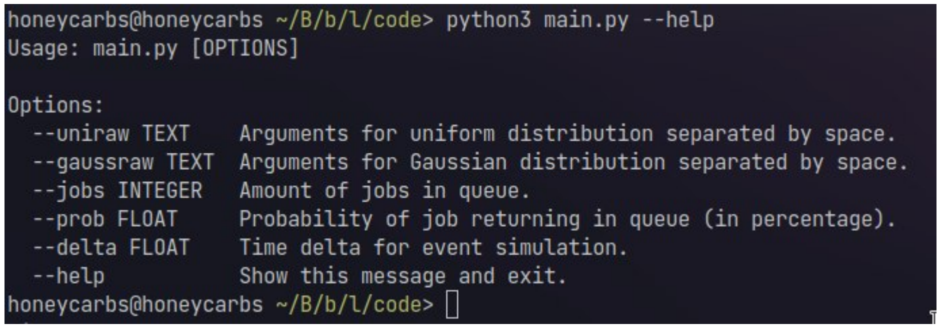


Рис. 3.2: Консольный интерфейс приложения



```
honeycarbs@honeycarbs ~/B/b/l/code> python3 main.py --help
Usage: main.py [OPTIONS]

Options:
  --uniraw TEXT    Arguments for uniform distribution separated by space.
  --gaussraw TEXT  Arguments for Gaussian distribution separated by space.
  --jobs INTEGER   Amount of jobs in queue.
  --prob FLOAT     Probability of job returning in queue (in percentage).
  --delta FLOAT    Time delta for event simulation.
  --help           Show this message and exit.
honeycarbs@honeycarbs ~/B/b/l/code>
```

Рис. 3.3: Информация об аргументах командной строки

Результат работы программы представлен на рисунках 3.1 и 3.2 – результаты событийного моделирования выделены синим цветом, а пошагового – красным.

3.2 Результаты работы

В таблице приведены результаты работы программы. Параметры: $\mathcal{N}(0, 1.0)$, $\mathcal{U}[1, 10]$, $\Delta = 0.1$.

Таблица 3.1: Результаты работы программы

модель	*процент повторе-ний*	*максимальное количество заявок в буфере*	*обработано заявок*	*количество заявок, отправленных назад в буфер*
Событийная	0	1	1000	0
Пошаговая		1	1000	0
Событийная	10	1	1113	113
Пошаговая		1	1103	103
Событийная	40	2	1673	673
Пошаговая		1	1713	713
Событийная	80	2	4879	3879
Пошаговая		2	5086	4086

4. Программный код

Для реализации программы был выбран язык Python. На листинге 4.1 представлена реализация пошагового метода.

Листинг 4.1: Релаизация пошагового метода

```
def TimekeepingSimulation(distribution, jobProcessor, jobsAmount=0, returnPercentage=0, step=0.001):
    processedJobs = 0
    currentTime = step
    generationTime = distribution.GetRandomValue()
    processTime = 0
    currentQueueLen = maxQueueLen = 0
    generatedJobs = 0
    returnedJobs = 0

    while processedJobs < jobsAmount + returnedJobs:

        if currentTime > generationTime and generatedJobs <= jobsAmount:
            currentQueueLen += 1
            generatedJobs += 1
            if currentQueueLen > maxQueueLen:
                maxQueueLen = currentQueueLen
                generationTime += distribution.GetRandomValue()
        if currentTime > processTime:
            if currentQueueLen > 0:
            processedJobs += 1

            if random.randint(1, 100) <= returnPercentage:
                returnedJobs += 1
                currentQueueLen += 1

            currentQueueLen -= 1
            processTime += jobProcessor.GetRandomValue()
        currentTime += step
    return maxQueueLen, processedJobs, returnedJobs
```

На листинге 4.2 представлена реализация событийного метода.

Листинг 4.2: Релаизация событийного метода

```
def EventSimulation(distribution, jobProcessor, jobsAmount=0,
   returnPercentage=0):
    processedJobs = 0
    currentQueueLen = 0
    maxQueueLen = 0

    jobs = [[distribution.GetRandomValue(), 'g']]

    free, isProcessed = True, False

    generatedJobs = 0
    returnedJobs = 0

    while processedJobs < jobsAmount + returnedJobs:

        job = jobs.pop(0)

        if job[1] == 'g' and generatedJobs <= jobsAmount:

            currentQueueLen += 1
            generatedJobs += 1

            if currentQueueLen > maxQueueLen:
                maxQueueLen = currentQueueLen

            enqueueJob(jobs, [job[0] +
               distribution.GetRandomValue(), 'g'])

            if free:
                isProcessed = True

        elif job[1] == 'p':

            processedJobs += 1

```

```
            if random.randint(1, 100) <= returnPercentage:
                returnedJobs += 1
                currentQueueLen += 1

            isProcessed = True

        if isProcessed:

            if currentQueueLen > 0:
                currentQueueLen -= 1
                t = jobProcessor.GetRandomValue()
                enqueueJob(jobs, [job[0] + t, 'p'])
                free = False
            else:
                free = True
            isProcessed = False

    return maxQueueLen, processedJobs, returnedJobs
```

На листинге 4.3 представлена функция помещения заявки в буфер.

Листинг 4.3: Вспомогательная функция помещения заявки в буфер

```
def enqueueJob(jobs, job):
    i = 0
    while i < len(jobs) and jobs[i][0] < job[0]:
        i += 1
    if 0 < i < len(jobs):
        jobs.insert(i - 1, job)
    else:
        jobs.insert(i, job)
```

На листинге 4.4 представлены функции распределений: равномерного и нормального.

Листинг 4.4: Функции распределений

```
class UniformDistributon:
    def __init__(self, lo: float, hi: float):
        self.lo = lo
        self.hi = hi

def GetRandomValue(self):
    return self.lo + (self.hi - self.lo) * random.random()

    class GaussianDistributon:
    def __init__(self, mu: float, sigma: float):
        self.mu = mu
        self.sigma = sigma

    def GetRandomValue(self):
        return normal(self.mu, self.sigma)
```