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 курсу: «Моделирование»

Тема: «Обслуживающий аппарат»

Студент: Ле Ни Куанг

Группа: ИУ7И-76Б

Преподаватель: Рудаков И. В.

Москва

2021

Содержание

1 Задание

Смоделировать систему, состоящую из генератора, очереди и обслуживающего аппарата. Закон генерации заявок выбирается равномерный. Закон в ОА берется из лабораторной работы №1. Определить оптимальную длину очереди, т.е. ту длину, при которой ни одно сообщение не исчезает. Должна быть возможность возвращения заявки в очередь после ее обработки с заданной вероятностью.

2 Теоритическая часть

2.1 Δt принцип

Принцип Δt заключается в последовательном анализе состояний всех блоков в момент $t + \Delta t$ по заданному состоянию блоков в момент 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.

Достоинство: равномерная протяжка времени.

Недостаток: значительные затраты машинного времени на реализацию моделирования системы. А при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, что исключает возможность получения адекватных результатов при моделировании.

2.2 Событийный принцип

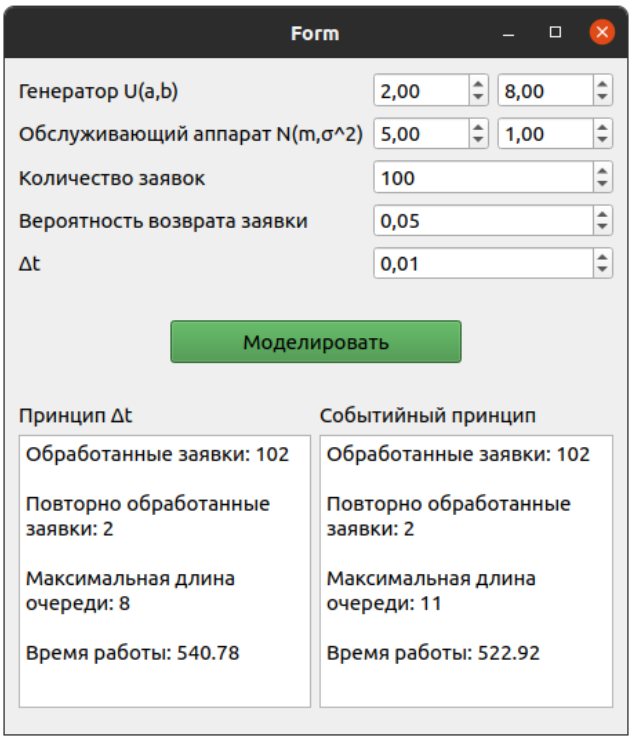
Характерное свойство моделируемых систем – состояние отдельных устройств изменяется в дискретные моменты времени, которые совпадают с моментами поступления сообщений в систему, моментами окончания решения задач, моментами возникающих аварийных сигналов и т.д. Поэтому, моделирование и продвижение текущего времени в системе удобно проводить использую событийный принцип, при котором состояние всех блоков системы анализируется лишь в момент наступления какого-либо события.

Момент наступления следующего события определяется минимальным значением из списка будущих событий, представляющих собой совокупность моментов ближайшего изменения состояний каждого из блоков системы.

Достоинство: не пропустим ни одного события

Недостаток: при большом количестве событий (сложная система) список необходимо просматривать постоянно (можно держать сортированным).

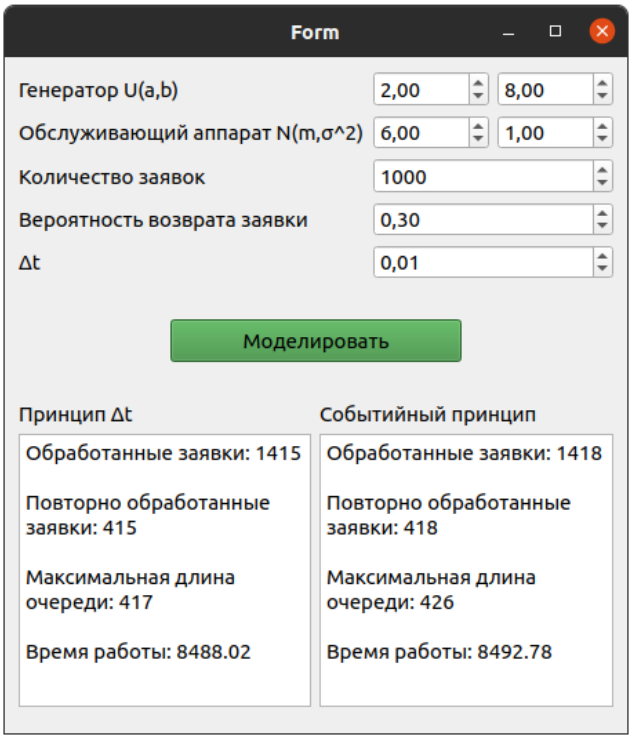
3 Результаты




```
Принцип Δt
Processed   0 tasks, t = 0.0000
Processed  20 tasks, t = 115.4200
Processed  40 tasks, t = 218.2100
Processed  60 tasks, t = 320.5900
Processed  80 tasks, t = 429.4800
Processed 100 tasks, t = 532.1600

Событийный принцип
Processed    0 tasks, t = 0.0000
Processed   20 tasks, t = 103.3627
Processed   40 tasks, t = 202.6244
Processed   60 tasks, t = 301.6904
Processed   80 tasks, t = 405.6612
Processed  100 tasks, t = 513.7761
```

Рисунок 1 – Среднее время между поступлением задачи = среднее время обработки задачи, вероятность возврата небольшой, размер очереди немного увеличивается




```
Принцип Δt
Processed   0 tasks, t = 0.0000
Processed 200 tasks, t = 1207.8800
Processed 400 tasks, t = 2411.0000
Processed 600 tasks, t = 3599.9400
Processed 800 tasks, t = 4795.2200
Processed 1000 tasks, t = 6007.8300
Processed 1200 tasks, t = 7199.5900
Processed 1400 tasks, t = 8395.3800

Событийный принцип
Processed    0 tasks, t = 0.0000
Processed  200 tasks, t = 1194.4176
Processed  400 tasks, t = 2417.7982
Processed  600 tasks, t = 3622.6552
Processed  800 tasks, t = 4799.1982
Processed 1000 tasks, t = 5990.0166
Processed 1200 tasks, t = 7180.6898
Processed 1400 tasks, t = 8377.8457
```

Рисунок 2 – Среднее время между поступлением задачи < среднее время обработки задачи, вероятность возврата = 0.3, размер очереди значительно увеличивается.

4 Листинг кода

Листинг 1 – Программная реализация обслуживающего аппарата

```
from .generator import RequestGenerator
from .queue import Queue
from .service import Service


LOG_FREQ = 5


class Model:
    def __init__(self, generator: RequestGenerator, queue: Queue,
        service: Service):
        self._generator = generator
        self._queue = queue
        self._service = service

    def time_based(self, n_tasks, dt=10e-3):
        generator, queue, service = self._generator, self._queue,
            self._service
        t_current = 0
        n_tasks_for_logging = n_tasks // LOG_FREQ

        while generator.generated_tasks < n_tasks \
                or not queue.is_empty or not service.is_ready:
            generator.elapse(dt)
            _, return_task = service.elapse(dt)

            if return_task:
                queue.enqueue(return_task)

            if generator.is_ready:
                task = generator.pop()
                queue.enqueue(task)

            if service.is_ready and not queue.is_empty:
                if service.completed_tasks % n_tasks_for_logging == 0:
                    print(f'Processed {service.completed_tasks:3d} tasks,
                        t = {t_current:.4f}')
                task = queue.dequeue()
                service.process(task)

            t_current += dt

        return t_current, service.completed_tasks, queue.len_max
```

```
    def event_based(self, n_tasks):
        """
        events: task_generated, task_completed
        """
        generator, queue, service = self._generator, self._queue,
            self._service
        t_current = 0
        n_tasks_for_logging = n_tasks // LOG_FREQ
        dt = 0

        while generator.generated_tasks < n_tasks \
                or not queue.is_empty or not service.is_ready:
            t_current += dt
            t_remain_task_generated = generator.elapse(dt)
            t_remain_task_completed, return_task = service.elapse(dt)

            if return_task:
                queue.enqueue(return_task)

            if generator.is_ready:
                task = generator.pop()
                queue.enqueue(task)

            if service.is_ready and not queue.is_empty:
                if service.completed_tasks % n_tasks_for_logging == 0:
                    print(f'Processed {service.completed_tasks:4d} tasks,
                        t = {t_current:.4f}')

                task = queue.dequeue()
                t_remain_task_completed = service.process(task)

            t_list = [t_remain_task_generated, t_remain_task_completed]
            t_list = list(filter(lambda x: x > 0, t_list))
            dt = min(t_list) if t_list else 0

        return t_current, service.completed_tasks, queue.len_max
```