Министерство науки и высшего образования Российской ФедерацииФедеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имениН.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 №6 по курсу «Моделирование»

Тема Моделирование сдачи экзамена

Студент Климов И.С.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва, 2022 г.

Задание

Создать модель для собственного объекта и промоделировать обработку запросов. В качестве объекта выбрана сдача экзамена на кафедре ИУ7.

Студенты приходят на экзамен через интервалы времени 3 ± 1 минуты. Если у студента сданы все лабораторные работы, то он становится в очередь на сдачу самого экзамена. Экзамен сдается за 17 ± 3 минуты, и вероятность успешной сдачи равна 0.8. В случае несдачи студент получает неудовлетворительную оценку и отправляется на пересдачу в другой день. Если же студент имеет задолженности по лабораторным, то он попадает в очередь на их защиту. Прием ведут два преподавателя: первый принимает за 6 ± 2 минуты (вероятность того, что лабораторные будут зачтены – 0.9), второй – за 8 ± 2 минуты (вероятность – 0.6). При зачете студент идет в очередь на экзамен, иначе – исправляет работы и снова становится в ту же очередь. Промоделировать процесс сдачи экзамена для 150 студентов. Определить вероятность сдачи.

Схемы модели

Структурная схема модели

На рисунке 1.1 представлена структурная схема модели.

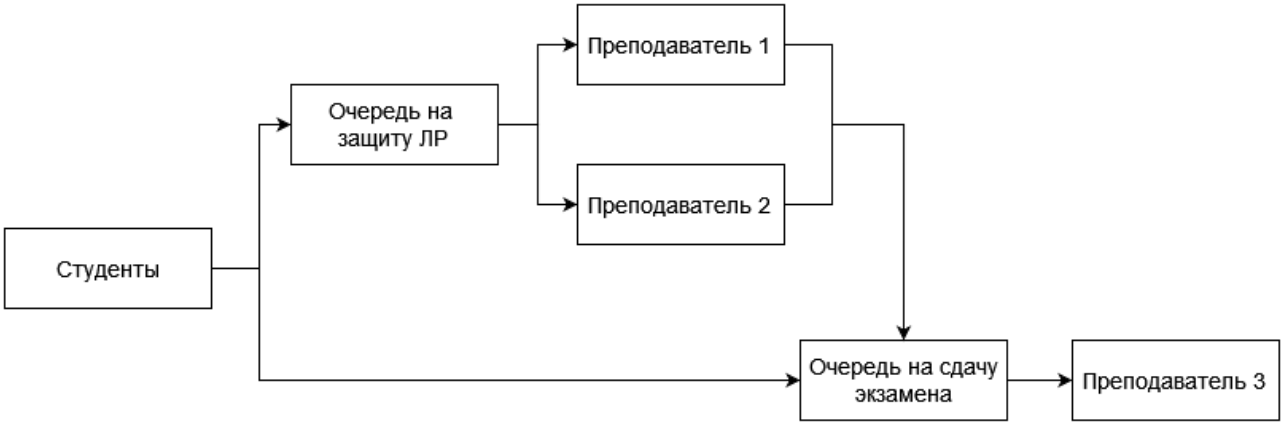

Рисунок 1.1 - Структурная схема модели

Схема модели в терминах СМО

Схема модели в терминах систем массового обслуживания представлена на рисунке 1.2.

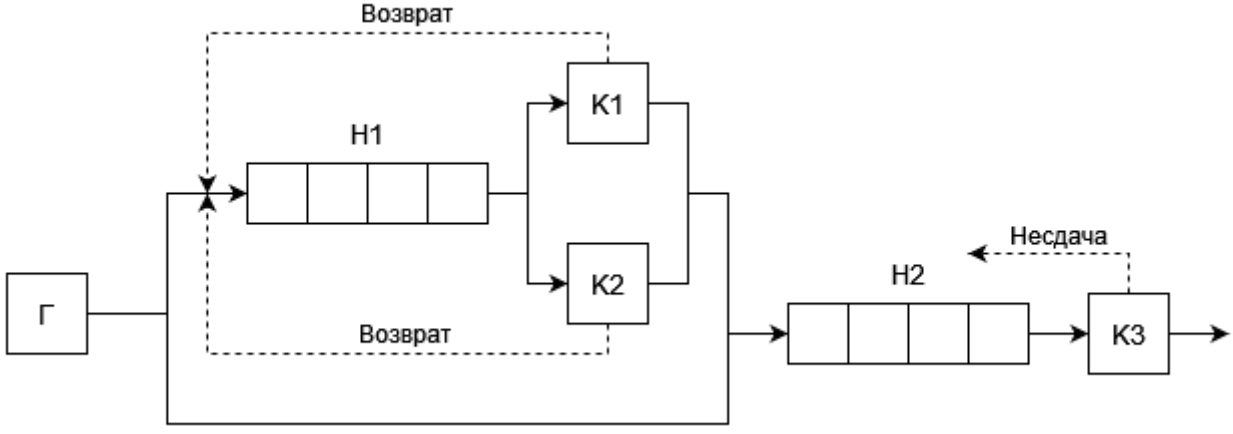

Рисунок 1.2 - Схема модели в терминах СМО

Согласно условию, каждое время сдачи (лабораторных и экзамена) подчиняется закону равномерного распределения.

Эндогенные переменные:

- время приема лабораторных i-ым преподавателем ($i = \overline{0;1}$);
- вероятность сдачи лабораторных i-ому преподавателю ($i = \overline{0;1}$);
- время приема экзамена;
- вероятность успешной сдачи экзамен.

Экзогенные переменные:

- $n_0 =$ числу студентов, сдавших экзамен;
- $n_1 =$ числу студентов, не сдавших экзамен.

Уравнения модели – $\frac{n_1}{n_0+n_1}$ (вероятность сдачи).

За единицу дискретного времени выбрана 0.01 минуты.

Текст программы

Ниже представлен текст программы, написанной на языке программирования Python.

```
import abc
import random
import queue


class Request:
    def __init__(self, create_time: float):
        self.__create_time = create_time
        self.__serve_time = -1.

    def get_creation_time(self) -> float:
        return self.__create_time

    def get_serve_time(self) -> float:
        return self.__serve_time

    def set_serve_time(self, value: float):
        self.__serve_time = value


class Generator:
    def __init__(self, a: float, b: float):
        self.__a = a
        self.__b = b
        self.__generated_time = 0.

    def is_ready(self, current_time: float) -> bool:
        return current_time >= self.__generated_time

    def update_generated_time(self):
        ti = self.__a + (self.__b - self.__a) * random.random()
        self.__generated_time += ti

    def create_request(self) -> Request:
        request = Request(self.__generated_time)
        self.update_generated_time()
        return request

    def set_generated_time(self, value: float):
        self.__generated_time = value


class Service:
    def __init__(self, a: float, b: float, probability: float):
        self.__a = a
        self.__b = b
```

```
        self.__free_time = 0.
        self.__pass_probability = probability

    def is_free(self, current_time: float) -> bool:
        return current_time >= self.__free_time

    def update_free_time(self, current_time: float):
        ti = self.__a + (self.__b - self.__a) * random.random()
        self.__free_time = current_time + ti

    def serve(self, request: Request, current_time: float) -> bool:
        self.update_free_time(current_time)

        if random.random() <= self.__pass_probability:
            request.set_serve_time(self.__free_time)
            return True

        return False

    def set_free_time(self, value: float):
        self.__free_time = value

    def get_free_time(self) -> float:
        return self.__free_time


class RequestQueue(queue.Queue):
    def __init__(self):
        super().__init__()
        self.__peak_len = 0

    def push(self, request: Request) -> bool:
        super().put(request)

        if self.qsize() > self.__peak_len:
            self.__peak_len = self.qsize()
            return True

        return False

    def pop(self):
        return super().get()

    def get_peak_len(self) -> int:
        return self.__peak_len

    def set_peak_len(self, value: int):
        self.__peak_len = value


class EventModel:
    class BaseEvent(metaclass=abc.ABCMeta):
        def __init__(self, current_time: float):
            self._current_time = current_time
```

```
        def get_current_time(self) -> float:
            return self._current_time

        @abc.abstractmethod
        def handle(self, model: 'EventModel'): ...

    class StudentPassedExamEvent(BaseEvent):
        def __init__(self, current_time: float):
            super().__init__(current_time)

        def handle(self, model: 'EventModel'):
            teacher = model.get_exam_teacher()
            exam_queue = model.get_exam_queue()

            if exam_queue.qsize() != 0 and teacher.is_free(self.get_current_time()):
                student = exam_queue.pop()
                passed = teacher.serve(student, self.get_current_time())

                if passed:
                    model.inc_passed_n()
                else:
                    model.inc_refused_n()


model.add_event(EventModel.StudentPassedExamEvent(teacher.get_free_time()))

    class StudentPassedLabsEvent(BaseEvent):
        def __init__(self, current_time: float, teacher_id: int):
            super().__init__(current_time)
            self.__teacher_id = teacher_id

        def handle(self, model: 'EventModel'):
            teacher = model.get_labs_teachers()[self.__teacher_id]

            if model.get_labs_queue().qsize() != 0:
                student: Request = model.get_labs_queue().pop()
                passed = teacher.serve(student, self.get_current_time())

                if passed:
                    student = Request(student.get_serve_time())
                    model.get_exam_queue().push(student)

                    if teacher.is_free(student.get_creation_time()):
                        model.add_event(EventModel.StudentPassedLabsEvent(
                            student.get_creation_time(), self.__teacher_id
                        ))
                else:
                    model.get_labs_queue().push(student)
                    model.add_event(EventModel.StudentPassedLabsEvent(
                        teacher.get_free_time(), self.__teacher_id)
                    )
```

```
    class StudentCameEvent(BaseEvent):
        def __init__(self, student: Request):
            super().__init__(student.get_creation_time())
            self.__student = student

        def handle(self, model: 'EventModel'):
            model.inc_students_n()

            if model.get_students_n() < model.get_max_students_n():
                next_student = model.get_students().create_request()
                model.add_event(EventModel.StudentCameEvent(next_student))

            ready_to_exam = random.choice([True, False])
            if ready_to_exam:
                model.get_exam_queue().push(self.__student)

                if model.get_exam_teacher().is_free(self.get_current_time()):
                    model.add_event(EventModel.StudentPassedExamEvent(
                        self.get_current_time()
                    ))
            else:
                model.get_labs_queue().push(self.__student)

                labs_teachers = model.get_labs_teachers()
                is_teachers_free = [labs_teachers[i].is_free(self.get_current_time())
                                    for i in range(2)]

                if is_teachers_free[0] and is_teachers_free[1]:
                    model.add_event(EventModel.StudentPassedLabsEvent(
                        self.get_current_time(), random.choice([0, 1])
                    ))
                elif is_teachers_free[0]:
                    model.add_event(EventModel.StudentPassedLabsEvent(
                        self.get_current_time(), 0
                    ))
                elif is_teachers_free[1]:
                    model.add_event(EventModel.StudentPassedLabsEvent(
                        self.get_current_time(), 1
                    ))

    def __init__(self, students: Generator, labs_teachers: list[Service],
                 exam_teacher: Service, max_students: int = 150):
        self.__students = students
        self.__labs_teachers = labs_teachers
        self.__exam_teacher = exam_teacher
        self.__labs_queue = RequestQueue()
        self.__exam_queue = RequestQueue()

        self.__max_students_n = max_students
        self.__end_time = 0.
        self.__refused_n = 0
        self.__passed_n = 0
        self.__students_n = 0
```

```
        self.__events: list['EventModel.BaseEvent'] = []

    def inc_students_n(self):
        self.__students_n += 1

    def inc_refused_n(self):
        self.__refused_n += 1

    def inc_passed_n(self):
        self.__passed_n += 1

    def get_students_n(self) -> int:
        return self.__students_n

    def get_max_students_n(self) -> int:
        return self.__max_students_n

    def get_students(self) -> Generator:
        return self.__students

    def get_labs_teachers(self) -> list[Service]:
        return self.__labs_teachers

    def get_exam_teacher(self) -> Service:
        return self.__exam_teacher

    def get_labs_queue(self) -> RequestQueue:
        return self.__labs_queue

    def get_exam_queue(self) -> RequestQueue:
        return self.__exam_queue

    def add_event(self, event: 'EventModel.BaseEvent'):
        self.__events.append(event)
        self.__events.sort(key=lambda x: x.get_current_time())

    def reset(self, max_students: int = 150):
        self.__students.set_generated_time(0.)

        for teacher in self.__labs_teachers:
            teacher.set_free_time(0.)
        self.__exam_teacher.set_free_time(0.)
        self.__labs_queue.set_peak_len(0)
        self.__exam_queue.set_peak_len(0)

        self.__max_students_n = max_students
        self.__end_time = 0.
        self.__refused_n = 0
        self.__passed_n = 0
        self.__students_n = 0

        first_student = self.__students.create_request()
        self.__events = [EventModel.StudentCameEvent(first_student)]
```

```
    def run(self) -> list[str]:
        self.reset()

        while self.__events:
            event = self.__events[0]
            self.__events.pop(0)

            self.__end_time = event.get_current_time()
            event.handle(self)

        return [
            str(self.__passed_n),
            str(self.__refused_n),
            f'{self.__passed_n / (self.__refused_n + self.__passed_n):.4f}',
            f'{self.__end_time:.2f}',
        ]
```

Результат

В результате разработана программа, позволяющая промоделировать процесс сдачи экзамена для 150 студентов согласно условию задачи. Результат работы программы представлен на рисунке 2.1.

Сдали экзамен	Не сдали экзамен	Вероятность сдачи	Время моделирования
123	27	0.8200	2564.81
115	35	0.7667	2542.16
129	21	0.8600	2574.96
116	34	0.7733	2588.64
125	25	0.8333	2569.85
115	35	0.7667	2603.64
118	32	0.7867	2547.50
123	27	0.8200	2569.86
113	37	0.7533	2551.57
126	24	0.8400	2561.37
128	22	0.8533	2516.08
121	29	0.8067	2572.06
117	33	0.7800	2524.37
117	33	0.7800	2564.06
120	30	0.8000	2516.89
120	30	0.8000	2562.64
122	28	0.8133	2563.88
120	30	0.8000	2577.66
113	37	0.7533	2530.57
124	26	0.8267	2548.28

Получить результат

Рисунок 2.1 - Результат работы программы