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 №5 по курсу «Моделирование»

Тема Моделирование информационного центра

Студент Климов И.С.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва, 2022 г.

Задание

В информационный центр приходят клиенты через интервалы времени 10 ± 2 минуты. Если все три, имеющихся оператора, заняты, клиенту отказывают в обслуживании. Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса за 20 ± 5, 40 ± 10, 40 ± 20 минут. Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные запросы сдаются в приемные накопители, откуда они выбираются для обработки. На первый компьютер – запросы от первого и второго операторов, на второй – от третьего. Время обработки на первом и втором компьютере равны 15 и 30 минутам соответственно. Промоделировать процесс обработки 300 запросов. Определить вероятность отказа.

Схемы модели

Структурная схема модели

На рисунке 1.1 представлена структурная схема модели.

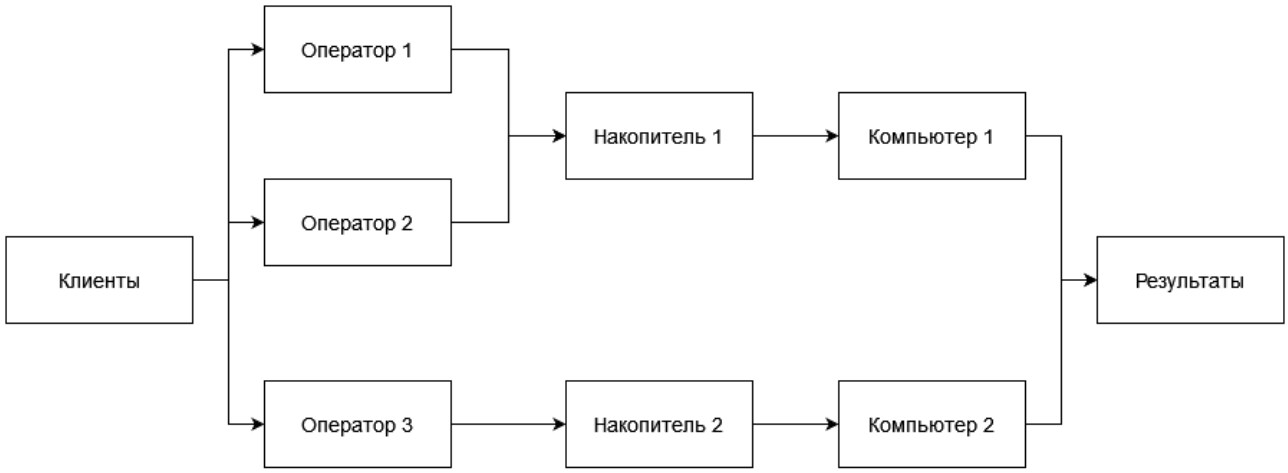

Рисунок 1.1 - Структурная схема модели

Схема модели в терминах СМО

В процессе взаимодействия клиентов с информационным центром возможны:

- режим нормального обслуживания (клиент выбирает одного из свободных операторов, но предпочитает того, у которого меньше номер);
- режим отказа (все операторы заняты).

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

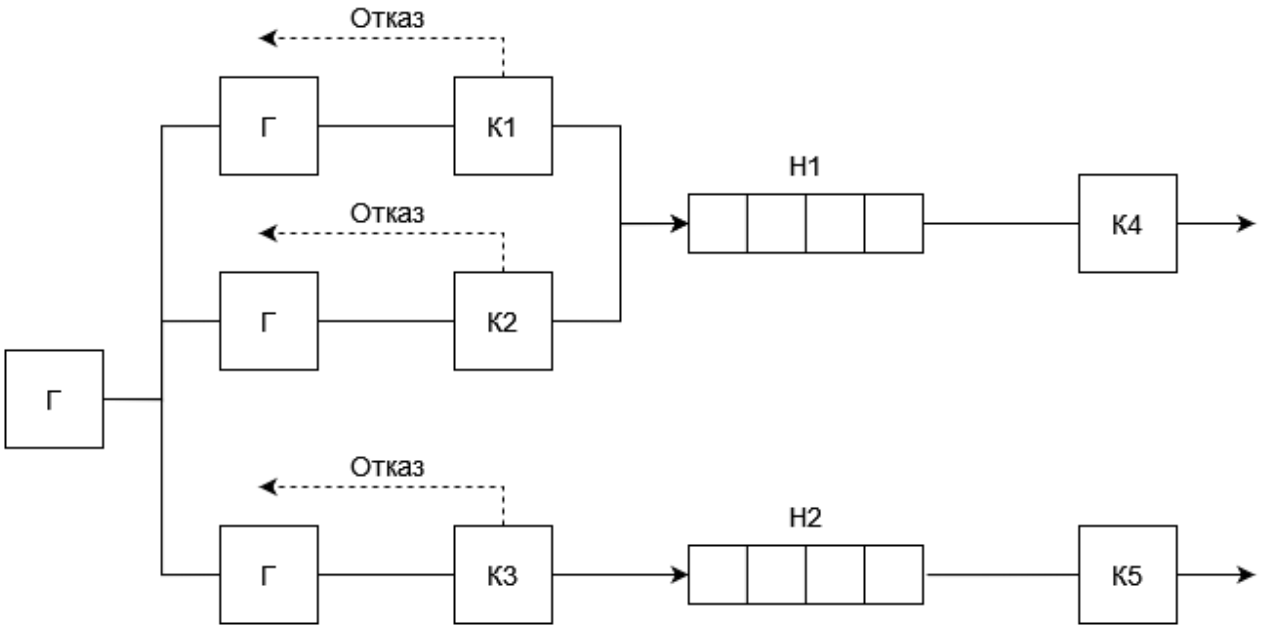

Рисунок 1.2 - Схема модели в терминах СМО

Согласно условию, время обработки заявки оператором подчиняется закону равномерного распределения, компьютер же выполняет каждую обработку за фиксированное время.

Эндогенные переменные:

- время обработки заданий i-ым оператором ($i = \overline{0;2}$);
- время решения задания на j-ом компьютере ($j = \overline{0;1}$).

Экзогенные переменные:

- $n_0 =$ числу обслуженных клиентов;
- $n_1 =$ числу клиентов получивших отказ.

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

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):
        self.__a = a
        self.__b = b
```

```
        self.__free_time = 0.

    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):
        self.update_free_time(current_time)
        request.set_serve_time(self.__free_time)

    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 EClientRequestServed(BaseEvent):
    def __init__(self, current_time: float, storage_id: int):
        super().__init__(current_time)
        self.__id = storage_id

    def handle(self, model: 'EventModel'):
        if model.get_storages()[self.__id].qsize() != 0:
            request = model.get_storages()[self.__id].pop()
            model.get_computers()[self.__id].serve(request, self._current_time)
            model.add_event(
                EventModel.EClientRequestServed(
                    model.get_computers()[self.__id].get_free_time(), self.__id
                )
            )

class EClientServed(BaseEvent):
    def __init__(self, request: Request, operator_id: int):
        super().__init__(request.get_creation_time())
        self.__client = request
        self.__operator_id = operator_id

    def handle(self, model: 'EventModel'):
        storage_id = 1 if self.__operator_id == 2 else 0
        model.get_storages()[storage_id].push(self.__client)

        if model.get_computers()[storage_id].is_free(self._current_time):
            model.add_event(EventModel.EClientRequestServed(
                self._current_time, storage_id)
            )

class EClientCame(BaseEvent):
    def __init__(self, request: Request):
        super().__init__(request.get_creation_time())
        self.__client = request

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

        if model.get_clients_n() < model.get_max_clients_n():
            next_client = model.get_generator().create_request()
            model.add_event(EventModel.EClientCame(next_client))

        for i, operator in enumerate(model.get_operators()):
            if operator.is_free(self._current_time):
                operator.serve(self.__client, self._current_time)
                request = Request(self.__client.get_serve_time())
                model.add_event(EventModel.EClientServed(request, i))

                model.inc_served_n()
                return

        model.inc_refused_n()
```

```
    def __init__(self, generator: Generator, operators: list[Service],
                 computers: list[Service], max_clients: int = 300):
        self.__generator = generator
        self.__operators = operators
        self.__computers = computers
        self.__storages = (RequestQueue(), RequestQueue())

        self.__max_clients_n = max_clients
        self.__end_time = 0.
        self.__refused_n = 0
        self.__served_n = 0
        self.__clients_n = 0

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

    def inc_clients_n(self):
        self.__clients_n += 1

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

    def inc_served_n(self):
        self.__served_n += 1

    def get_clients_n(self) -> int:
        return self.__clients_n

    def get_max_clients_n(self) -> int:
        return self.__max_clients_n

    def get_generator(self) -> Generator:
        return self.__generator

    def get_operators(self) -> list[Service]:
        return self.__operators

    def get_storages(self) -> tuple[RequestQueue, RequestQueue]:
        return self.__storages

    def get_computers(self) -> list[Service]:
        return self.__computers

    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_clients: int = 300):
        self.__generator.set_generated_time(0.)

        for operator in self.__operators:
            operator.set_free_time(0.)

        for computer in self.__computers:
            computer.set_free_time(0.)
```

```
        for storage in self.__storages:
            storage.set_peak_len(0)

        self.__max_clients_n = max_clients
        self.__end_time = 0.
        self.__refused_n = 0
        self.__served_n = 0
        self.__clients_n = 0

        first_client = self.__generator.create_request()
        self.__events = [EventModel.EClientCame(first_client)]

    def get_refuse_probability(self) -> float:
        self.reset()

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

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

        return self.__refused_n / (self.__refused_n + self.__served_n)

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

Результат

В результате разработана программа, позволяющая промоделировать процесс обработки 300 запросов согласно условию задачи. На рисунке 2.1 представлен результат выполнения.

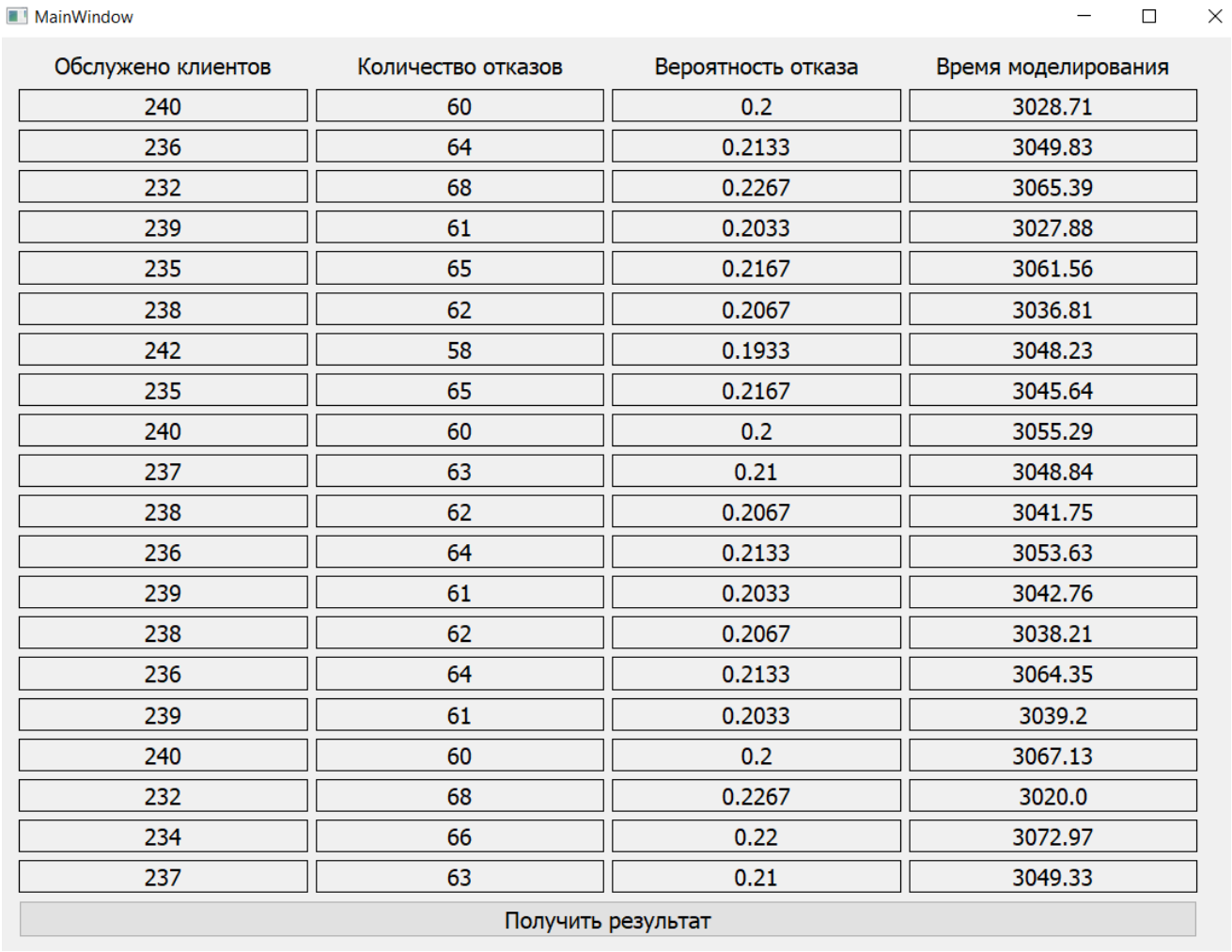

Рисунок 2.1 - Результат работы прораммы