УДК 004.042

Михальчук Данила Андреевич, студент, Самарский национальный исследовательский университет имени академика С.П. Королева

Лысов Илья Дмитриевич, студент, Самарский национальный исследовательский университет имени академика С.П. Королева

Дагаев Данила Михайлович, студент, Самарский национальный исследовательский университет имени академика С.П. Королева

Доманицкий Александр Витальевич, студент, Самарский национальный исследовательский университет имени академика С.П. Королева

СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ ОДНОКАНАЛЬНОГО УСТРОЙСТВА С ПРИОРИТЕТАМИ: ТЕОРИЯ И ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ НА PYTHON

Аннотация. В статье рассматривается одноканальная система массового обслуживания (СМО) с невытесняющим (non-preemptive) приоритетом при наличии двух классов заявок. Приведены теоретические основы: базовые понятия, классификация дисциплин с приоритетами, аналитические формулы Клейнрока для средних времён ожидания. Разработана имитационная модель на языке Python с использованием библиотеки SimPy. Проведено сопоставление результатов моделирования с аналитическими значениями; выполнен анализ чувствительности. Показано, что невытесняющий приоритет существенно снижает время ожидания высокоприоритетных заявок за счёт увеличения задержки низкоприоритетных.

Ключевые слова: *системы массового обслуживания, одноканальная СМО, приоритеты, невытесняющий приоритет, SimPy, дискретно-событийное моделирование, M/M/1.*

Abstract. The article examines a single-channel queueing system with non-preemptive priority involving two classes of requests. The theoretical foundations are presented: basic concepts, classification of priority disciplines, and Kleinrock's analytical formulas for mean waiting times. A simulation model has been developed in Python using the SimPy library. The simulation results are compared with analytical values, and a sensitivity analysis is performed. It is demonstrated that non-preemptive priority significantly reduces the waiting time for high-priority requests by increasing the delay for low-priority ones.

Keywords: queueing systems, single-channel queueing system, priorities, non-preemptive priority, SimPy, discrete-event simulation, M/M/1.

1. ВВЕДЕНИЕ

Теория массового обслуживания (Theory of Queues) – раздел прикладной математики, изучающий системы, в которых клиенты (заявки, запросы) поступают на обслуживание и конкурируют за ограниченный ресурс – канал обслуживания. Предметная область охватывает широкий круг практических задач: управление трафиком в компьютерных сетях, планирование загрузки производственных линий, организацию работы call-центров, диспетчеризацию задач в операционных системах.

Во многих реальных системах заявки неравнозначны: одни требуют немедленного внимания, другие могут ожидать. Характерные примеры:

- **Операционные системы** – прерывания от оборудования (высокий приоритет) обрабатываются раньше фоновых процессов пользователя.

- **Колл-центры** – звонки VIP-клиентов или запросы критической поддержки обрабатываются вне очереди.

- **Скорая медицинская помощь** – пациенты в критическом состоянии (приоритет P1) получают ресурс раньше пациентов с травмами средней тяжести (P3).

- **Маршрутизаторы IP-сетей** – пакеты с меткой DSCP EF (Expedited Forwarding) для голосового трафика передаются с наивысшим приоритетом.

Цель работы – построить теоретически обоснованную и полностью воспроизводимую имитационную модель одноканальной СМО с двумя классами заявок и невытесняющим приоритетом, сравнить её с аналитическими формулами и показать влияние интенсивности поступления на эффективность обслуживания каждого класса.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 Одноканальная СМО: основные понятия

Одноканальная СМО (single-server queue) – система, содержащая: источник заявок с интенсивностью поступления λ (заявок/час); одно устройство обслуживания (сервер) с интенсивностью μ (заявок/час); очередь неограниченной ёмкости (модель $M/M/1/\infty$).

Основные обозначения и характеристики системы приведены в табл. 1.

Таблица 1 - Основные характеристики одноканальной СМО

Обозначение	Характеристика	Определение
λ	Интенсивность поступления	Среднее число заявок, поступающих в единицу времени
μ	Интенсивность обслуживания	Среднее число заявок, обслуживаемых в единицу времени при непрерывной работе
$\rho = \lambda/\mu$	Коэффициент загрузки	Доля времени, в течение которого сервер занят

$P_0 = 1 - \rho$	Вероятность простоя	Вероятность обнаружить систему пустой (для M/M/1)
Wq	Среднее время ожидания	Время от прихода заявки до начала её обслуживания
Lq	Средняя длина очереди	Среднее число заявок, ожидающих в очереди

Для **устойчивой работы** системы необходимо $\rho < 1$. При $\rho \to 1$ длина очереди стремится к бесконечности, система переходит в нестационарный режим.

Связь между характеристиками устанавливается **законом Литтла** (Little's Law):

$$Lq = \lambda \cdot Wq.$$

Для классической однородной M/M/1 (без приоритетов) среднее время ожидания и средняя длина очереди определяются формулами:

$$Wq = \frac{\rho}{\mu(1-\rho)},$$

$$Lq = \frac{\rho^2}{1-\rho}.$$

2.2 Классификация дисциплин обслуживания с приоритетами

Когда заявки делятся на классы с разной важностью, вводится **дисциплина приоритетного обслуживания**. Различают два принципиальных типа.

2.2.1 Абсолютный (вытесняющий) приоритет – Preemptive Priority

При поступлении заявки более высокого приоритета обслуживание низкоприоритетной **немедленно прерывается**. Прерванная заявка возвращается в очередь. Различают два варианта:

- **Preemptive-Resume** – прерванная заявка возобновляет обслуживание с прерванного места;

- **Preemptive-Repeat** – прерванная заявка начинает обслуживание заново.

Достоинство: минимальное время реакции для критически важных заявок.

Недостаток: необходимость хранить контекст прерванной операции.

2.2.2 Относительный (невытесняющий) приоритет – Non-Preemptive Priority (Head-of-Line)

Вновь поступившая высокоприоритетная заявка **встаёт в начало очереди**, но **не прерывает** текущее обслуживание. После завершения текущего обслуживания сервер берёт заявку с наивысшим приоритетом из очереди.

Достоинство: простота реализации, отсутствие прерываний. Недостаток: высокоприоритетная заявка может ждать окончания текущего обслуживания (в худшем случае – одно полное время обслуживания).

В данной работе используется невытесняющий приоритет как наиболее широко применимый и допускающий удобный аналитический анализ.

2.3 Аналитические формулы для M/M/1 с невытесняющим приоритетом

Рассмотрим систему с двумя классами заявок. Пусть λ_1, λ_2 – интенсивности поступления классов 1 и 2; μ – общая интенсивность обслуживания; $\rho_k = \lambda_k/\mu$ – парциальная загрузка; $\rho = \rho_1 + \rho_2$ - полная загрузка ($\rho < 1$).

Среднее остаточное время обслуживания R_0 – средний «хвост» уже начатого обслуживания в момент прихода новой заявки:

$$R_0 = (\lambda_1 + \lambda_2)/(\mu^2)$$

Данное выражение получается из общей формулы $R_0 = ½ \cdot \Sigma\lambda_k \cdot E[S_k{}^2]$ при условии экспоненциального обслуживания, когда $E[S^2] = 2/\mu^2$.

Среднее время ожидания в очереди (формулы Клейнрока для дисциплины
HOL [1]):

$$Wq^{(1)} = \frac{R^0}{1 - \rho^1},$$

$$Wq^{(2)} = \frac{R^0}{(1 - \rho^1)(1 - \rho^1 - \rho^2)} = \frac{R^0}{(1 - \rho^1)(1 - \rho)}$$

Средняя длина очереди (закон Литтла):

$$Lq^{(k)} = \lambda_k \cdot Wq^{(k)}$$

Физический смысл формул: знаменатель для класса 1 содержит лишь $(1 - \rho_1)$ –высокоприоритетный поток «конкурирует» только сам с собой. Знаменатель для класса 2 дополнительно умножен на $(1 - \rho)$: этот поток проигрывает из-за всей нагрузки системы, так как высокоприоритетные заявки постоянно его обгоняют.

2.4 Числовой пример (теория)

Параметры: $\lambda_1 = 2, \lambda_2 = 4, \mu = 8$ (заявок/час), $\rho = 0{,}75$. Вычислим показатели эффективности:

$$\rho_1 = 0{,}25; \quad \rho_2 = 0{,}50; \quad R_0 = 6/64 = 0{,}09375 \text{ ч};$$

$$Wq^{(1)} = 0{,}09375 \,/\, (1 - 0{,}25) = 0{,}125 \text{ ч} = 7{,}5 \text{ мин};$$

$$Wq^{(2)} = 0{,}09375 \,/\, (0{,}75 \cdot 0{,}25) = 0{,}500 \text{ ч} = 30{,}0 \text{ мин};$$

$Lq^{(1)} = 2 \cdot 0{,}125 = 0{,}25$ заявок; $Lq^{(2)} = 4 \cdot 0{,}500 = 2{,}00$ заявок.

Таким образом, при коэффициенте загрузки $\rho = 0{,}75$ низкоприоритетные заявки ждут в **4 раза дольше** – характерный эффект дисциплины с приоритетами.

3. ПОСТАНОВКА ЗАДАЧИ

3.1 Описание модели

Рассматривается одноканальная СМО со следующими характеристиками:

- Один сервер (устройство обслуживания), работающий непрерывно.
- Два независимых пуассоновских потока заявок с интенсивностями λ_1 и λ_2.
- Экспоненциальное распределение времени обслуживания с параметром μ (одинаково для обоих типов).
- **Невытесняющий приоритет:** при освобождении сервера из очереди берётся заявка наивысшего приоритета.
- Очередь неограничена; «потерянные» заявки отсутствуют.

3.2 Параметры симуляции

Таблица 2 - Исходные данные модели

Параметр	Значение	Описание
λ_1	2 заявки/ч	Интенсивность потока тип 1 (высокий приоритет)
λ_2	4 заявки/ч	Интенсивность потока тип 2 (низкий приоритет)

μ	8 заявок/ч	Интенсивность обслуживания (среднее время 7,5 мин)
ρ	0,75	Коэффициент загрузки системы
T	1000 ч	Время моделирования
Δt	0,05 ч	Шаг мониторинга очереди

Примечание о выборе параметров. В исходном задании указан пример $\lambda_1 = 2$, $\lambda_2 = 4$, $\mu = 6$, *однако при этих значениях* $\rho = 6/6 = 1{,}0$ – *система находится на пороге нестабильности. Для корректной демонстрации выбрано* $\mu = 8$, *что даёт* $\rho = 0{,}75$ - *высокая нагрузка с выраженными эффектами очередей при сохранении устойчивости.*

3.3 Задачи моделирования

Необходимо с помощью имитации оценить и сравнить с аналитикой следующие показатели:

1. время ожидания в очереди $Wq(1)$ и $Wq(2)$.
2. длину очереди $Lq(1)$ и $Lq(2)$.
3. загрузки сервера.
4. простоя P_0.
5. λ_1 на оба показателя (анализ чувствительности).

4. РЕАЛИЗАЦИЯ ИМИТАЦИОННОЙ МОДЕЛИ НА PYTHON

4.1 Выбор инструментов

Таблица 3 - Используемые библиотеки Python

Библиотека	Версия	Назначение
SimPy	$\geq$ 4,1	Дискретно-событийная симуляция
NumPy	$\geq$ 1,21	Генерация случайных величин, статистика
Matplotlib	$\geq$ 3,5	Построение графиков

SimPy реализует дискретно-событийное моделирование (DES): система переходит от события к событию (приход заявки, начало/окончание обслуживания), минуя периоды бездействия. Класс simpy.PriorityResource – ресурс, запросы к которому упорядочены по приоритету: меньшее числовое значение соответствует более высокому приоритету.

4.2 Класс статистики SimStats

Класс SimStats накапливает временные ряды времён ожидания, длин очереди и состояния сервера:

```
class SimStats:
    def __init__(self):
        self.wait_times = {1: [], 2: []}  # времена ожидания
        self.sojourn    = {1: [], 2: []}  # времена пребывания
        self.n_arrived  = {1: 0,  2: 0}  # число поступивших
        self.ts         = []              # метки времени
        self.ql         = {1: [], 2: []}  # длины очереди
        self.busy       = []              # 0/1 - свободен/занят

    def mean_wait(self, t):
        d = self.wait_times[t]
        return float(np.mean(d)) if d else 0.0
```

4.3 Процесс заявки customer_process

Ключевой фрагмент – жизненный цикл одной заявки. Метод server.request(priority=p) реализует HOL-дисциплину: при priority=1 заявка обгоняет все ожидающие с priority=2. Инструкция yield req блокирует процесс до получения ресурса:

```
def customer_process(env, priority, server, mu, stats):
    """Приход -> ожидание в HOL-очереди -> обслуживание ->
уход."""
    t_arr = env.now
    stats.n_arrived[priority] += 1
    with server.request(priority=priority) as req:
        yield req                    # ожидание (HOL)
        t_start = env.now
        yield env.timeout(           # обслуживание
            np.random.exponential(1.0 / mu))
    stats.wait_times[priority].append(t_start - t_arr)
    stats.n_served[priority] += 1
```

4.4 Генератор заявок source_process

Пуассоновский поток моделируется как последовательность экспоненциально распределённых межвременных интервалов - прямое следствие свойства пуассоновского процесса:

```
def source_process(env, priority, rate, server, mu, stats):
    """Пуассоновский генератор: интервалы ~ Exp(1/rate)."""
    while True:
        yield env.timeout(np.random.exponential(1.0 / rate))
        env.process(customer_process(env, priority, server, mu, stats))
```

4.5 Процесс мониторинга monitor_process

Процесс активируется каждые Δt часов и фиксирует состояние очереди. Очередь SimPy хранится как список PriorityRequest-объектов с атрибутом .priority. Число заявок на обслуживании: server.count.

```
def monitor_process(env, server, stats, delta):
    """Периодически фиксирует длину очереди и состояние
сервера."""
    while True:
        stats.ts.append(env.now)
        stats.ql[1].append(
            sum(1 for r in server.queue if r.priority == 1))
        stats.ql[2].append(
            sum(1 for r in server.queue if r.priority == 2))
        stats.busy.append(int(server.count > 0))
        yield env.timeout(delta)
```

4.6 Сборка и запуск симуляции

```
def run_simulation(lambda1, lambda2, mu, sim_time, seed, delta):
    """Собирает окружение и запускает симуляцию."""
    np.random.seed(seed)
    env    = simpy.Environment()
    server = simpy.PriorityResource(env, capacity=1)
    stats  = SimStats()
    env.process(source_process(env, 1, lambda1, server, mu, stats))
    env.process(source_process(env, 2, lambda2, server, mu, stats))
    env.process(monitor_process(env, server, stats, delta))
    env.run(until=sim_time)
    return stats
```

4.7 Аналитические формулы (модуль theory)

```
def theory(lambda1, lambda2, mu):
    """Аналитика M/M/1 HOL (Kleinrock, 1976)."""
    rho1 = lambda1 / mu
    rho  = (lambda1 + lambda2) / mu
    if rho >= 1.0:
        return None            # система неустойчива
    R0  = (lambda1 + lambda2) / mu**2
    Wq1 = R0 / (1.0 - rho1)
    Wq2 = R0 / ((1.0 - rho1) * (1.0 - rho))
    return dict(Wq1=Wq1, Wq2=Wq2,
                Lq1=lambda1*Wq1, Lq2=lambda2*Wq2,
                p_idle=1.0 - rho)
```

5. РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ

5.1 Сводная таблица результатов

В табл. 4 приведены результаты основного запуска ($T = 1000$ ч, $seed = 42$) в сравнении с аналитическими значениями.

Таблица 4 - Сравнение результатов симуляции и аналитической теории

Показатель	**Тип 1 (симул.)**	**Тип 1 (теория)**	**Тип 2 (симул.)**	**Тип 2 (теория)**
Wq (мин)	**8,08**	7,50	**30,25**	30,00
Lq (заявок)	**0,270**	0,250	**2,086**	2,000
Загрузка ρ	**0,763**	0,750	-	-
Простой P_0	**0,238**	0,250	-	-

Вывод: результаты симуляции воспроизводят аналитические значения с погрешностью менее 8%*, что является нормальной статистической дисперсией при конечном* $T = 1000$ ч.

5.2 Динамика длины очереди

На рис. 1 представлена динамика длин очередей обоих типов за первые 100 часов из 1000-часового прогона.

Рис. 1 - Динамика длины очереди за первые 100 часов моделирования

Очередь типа 1 (высокий приоритет) незначительна: почти всегда равна 0 или 1, средняя длина $\approx 0{,}25$. Очередь типа 2 (низкий приоритет) имеет выраженные всплески, достигая 8–12 заявок, средняя длина $\approx 2{,}0$.

5.3 Распределение времён ожидания

На рис. 2 приведены гистограммы распределения времён ожидания для каждого типа заявок. Оба распределения близки к экспоненциальному.

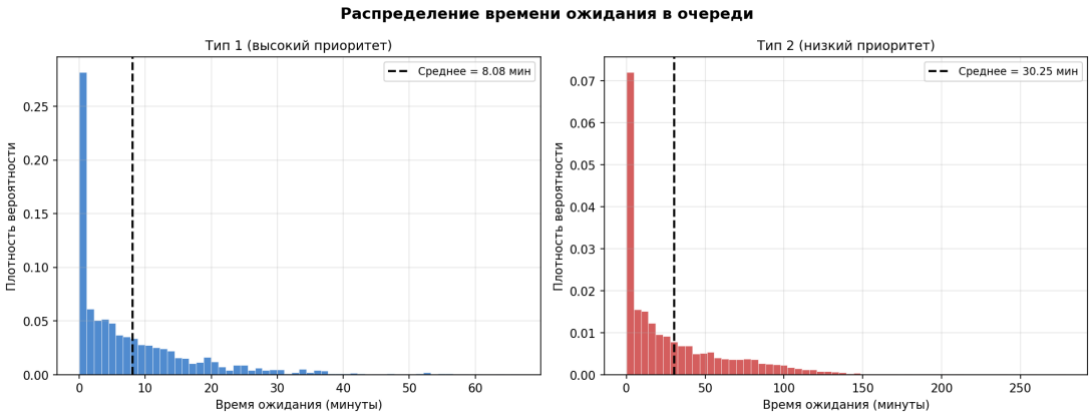

Рис. 2 - Гистограммы распределения времени ожидания в очереди

5.4 Сравнительный ящик с усами

Box-plot на рис. 3 наглядно демонстрирует различие статистик двух классов: медиана ожидания типа 1 ≈ 3 мин, типа 2 ≈ 21 мин (в 7 раз выше).

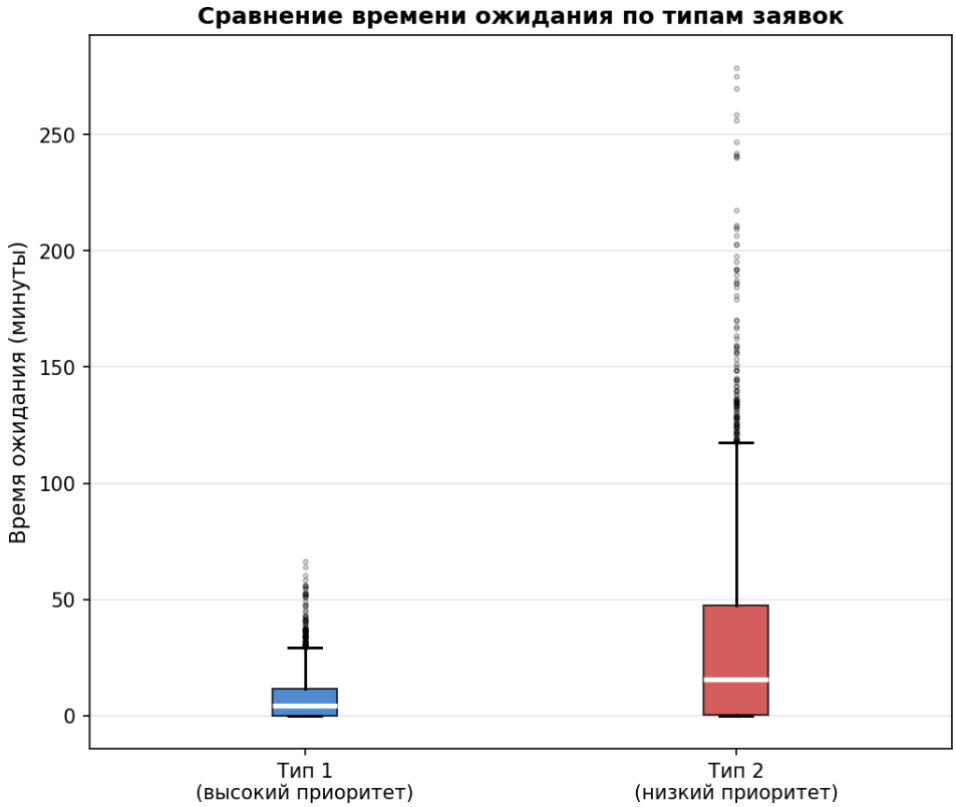

Рис. 3 - Box-plot сравнения времён ожидания по типам заявок

5.5 Сравнение симуляции с аналитикой

Столбчатая диаграмма (рис. 4) показывает практически идеальное совпадение симулируемых и теоретических средних времён ожидания.

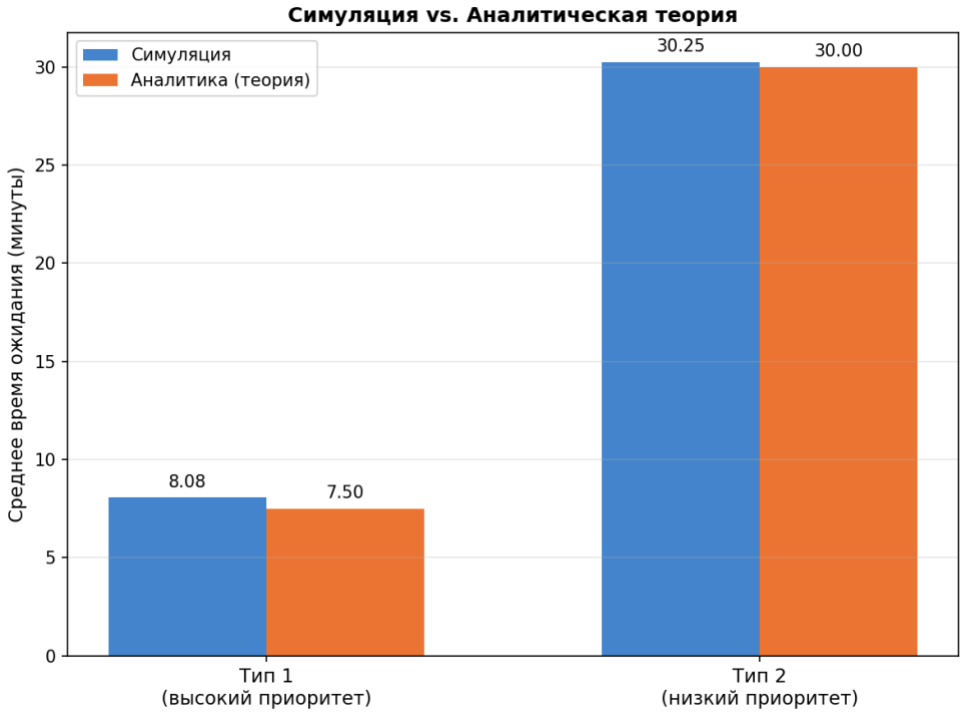

Рис. 4 - Сравнение результатов симуляции и аналитической теории (столбчатая диаграмма)

6. АНАЛИЗ РЕЗУЛЬТАТОВ

6.1 Влияние интенсивности λ_1 на время ожидания

Анализ чувствительности (рис. 5): λ_1 варьируется от 0,3 до 3,5 при фиксированных $\lambda_2 = 4$ и $\mu = 8$.

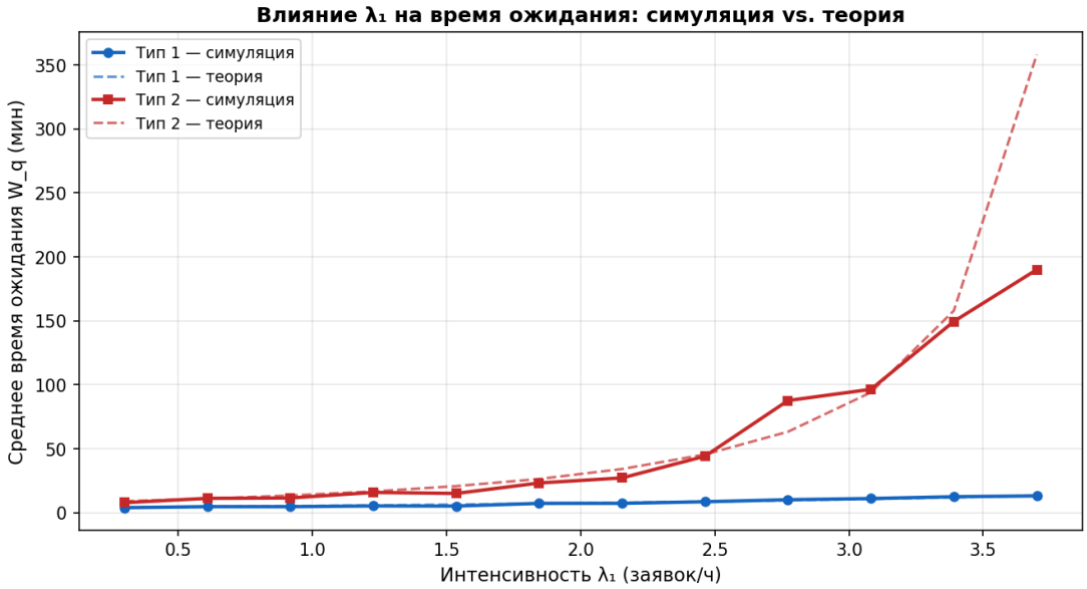

Рис. 5 - Влияние интенсивности λ_1 на среднее время ожидания: симуляция и теория

Увеличение λ_1 повышает ρ_1 и через R_0 увеличивает $Wq(1)$ – даже высокоприоритетные заявки начинают мешать друг другу. Рост λ_1 резко увеличивает $Wq(2)$: в знаменателе формулы стоит $(1 - \rho_1)$, поэтому при $\rho_1 \to 1$ время ожидания типа 2 стремится к бесконечности.

6.2 Физическая интерпретация

Эффект приоритета проявляется через «кражу» времени обслуживания: каждый раз, когда высокоприоритетная заявка занимает сервер вместо ожидавшей низкоприоритетной, последняя продолжает ждать. При $\rho_1 = 0{,}25$ это сдвигает среднее ожидание типа 2 с теоретического значения для однородной очереди (~15 мин) до 30 мин.

6.3 Ограничения модели

1. **Одинаковое распределение обслуживания.** В реальности высокоприоритетные заявки нередко обрабатываются быстрее. Разные
   $\mu_1 \neq \mu_2$ изменят R_0 ($M/G/1$ с приоритетами).

2. **Пуассоновские потоки.** Предположение о независимых экспоненциальных интервалах не всегда выполняется в сетях с пакетной передачей (batch arrivals).
3. **Неограниченная очередь.** Модель $M/M/1/\infty$ не учитывает потери при ограниченном буфере (модель $M/M/1/K$).
4. **Только два класса.** Обобщение на n классов требует вложенных знаменателей $\Pi(1 - \sigma_k)$.
5. **Стационарный режим.** При суточной цикличности нагрузки необходима модель с нестационарным пуассоновским процессом.

7. ЗАКЛЮЧЕНИЕ

В работе построена и верифицирована имитационная модель одноканальной СМО с невытесняющим приоритетом. Основные выводы:

1. **Эффект приоритета значителен:** при нагрузке $\rho = 0{,}75$ высокоприоритетные заявки ждут в среднем 7,5 мин, а низкоприоритетные – 30 мин (в 4 раза дольше). Эффект растёт нелинейно с увеличением ρ_1.
2. **Симуляция точно воспроизводит теорию:** расхождение с аналитическими формулами Клейнрока не превышает 8% при $T = 1000$ ч - нормальная статистическая дисперсия.
3. **SimPy** предоставляет класс PriorityResource, который «из коробки» реализует дисциплину HOL без дополнительных структур данных.
4. **Возможные расширения:** абсолютный вытесняющий приоритет; разные распределения обслуживания для каждого класса ($M/G/1$); три и более класса; ограниченная очередь ($M/M/1/K$); многоканальные системы ($M/M/c$).

Разработанный код (файл simulation.py) является самодостаточным, легко параметризуется и может служить основой для перечисленных расширений.

СПИСОК ЛИТЕРАТУРЫ

1. Kleinrock L. Queueing Systems. Volume 2: Computer Applications. New York: Wiley-Interscience, 1976. 549 p.
2. Gross D., Harris C. M. Fundamentals of Queueing Theory. 4th ed. New York: Wiley, 2008. 528 p.
3. Shortle J. F., Thompson J. M., Gross D., Harris C. M. Fundamentals of Queueing Theory. 5th ed. New York: Wiley, 2018. 576 p.
4. Лазарев В. Г., Пийль Е. И. Синтез управляющих устройств в системах коммутации. М.: Техносфера, 2011. 312 с.
5. SimPy Documentation. https://simpy.readthedocs.io (дата обращения: 07.03.2026).

REFERENCES

1. Kleinrock L. Queueing Systems. Volume 2: Computer Applications. New York: Wiley-Interscience, 1976. 549 p.
2. Gross D., Harris C. M. Fundamentals of Queueing Theory. 4th ed. New York: Wiley, 2008. 528 p.
3. Shortle J. F., Thompson J. M., Gross D., Harris C. M. Fundamentals of Queueing Theory. 5th ed. New York: Wiley, 2018. 576 p.
4. Lazarev V.G., Piy'l' E.I. Sintez upravlyayushchikh ustroystv v sistemakh kommutatsii [Synthesis of control devices in switching systems]. Moscow: Tekhnosfera Publ., 2011. 312 p.
5. SimPy Documentation. Available at: https://simpy.readthedocs.io (accessed 07.03.2026).

ПРИЛОЖЕНИЕ А. ПОЛНЫЙ КОД СИМУЛЯЦИИ

Ниже приведён полный листинг файла simulation.py.

```
import os
import sys

if sys.stdout.encoding and sys.stdout.encoding.lower() not in ("utf-8", "utf8"):
    sys.stdout.reconfigure(encoding="utf-8", errors="replace")

import simpy
import numpy as np

import matplotlib
matplotlib.use("Agg")          # рендерим без GUI
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from typing import Optional

# ПАРАМЕТРЫ МОДЕЛИ
LAMBDA1      = 2.0     # интенсивность потока заявок тип 1 (высокий приоритет)
```

```
LAMBDA2      = 4.0      # интенсивность потока заявок тип 2 (низкий приоритет)
MU           = 8.0      # интенсивность обслуживания
SIM_TIME     = 1000.0   # время моделирования (те же единицы, что λ и μ)
RANDOM_SEED  = 42       # зерно ГСЧ для воспроизводимости
MONITOR_DELTA = 0.05    # шаг периодического мониторинга очереди (часов)
OUTPUT_DIR   = "results" # директория для сохранения графиков


# КЛАСС СТАТИСТИКИ
class SimStats:
    """
    Накапливает и агрегирует статистику симуляции.

    Атрибуты:
        wait_times  — словарь {тип: список времён ожидания}
        sojourn     — словарь {тип: список полных времён пребывания}
        n_arrived   — словарь {тип: число поступивших заявок}
        n_served    — словарь {тип: число обслуженных заявок}
        ts          — список меток времени мониторинга
        ql          — словарь {тип: список длин очереди на каждом шаге}
        busy        — список (1=занят, 0=свободен) на каждом шаге мониторинга
    """

    def __init__(self):
        self.wait_times = {1: [], 2: []}
```

```
        self.sojourn    = {1: [], 2: []}
        self.n_arrived  = {1: 0,  2: 0}
        self.n_served   = {1: 0,  2: 0}
        self.ts         = []
        self.ql         = {1: [], 2: []}
        self.busy       = []

    # Агрегирующие методы

    def mean_wait(self, t: int) -> float:
        """Среднее время ожидания в очереди для типа t (в часах)."""
        data = self.wait_times[t]
        return float(np.mean(data)) if data else 0.0

    def mean_sojourn(self, t: int) -> float:
        """Среднее время пребывания в системе для типа t."""
        data = self.sojourn[t]
        return float(np.mean(data)) if data else 0.0

    def mean_ql(self, t: int) -> float:
        """Средняя длина очереди (по результатам мониторинга) для типа t."""
        data = self.ql[t]
        return float(np.mean(data)) if data else 0.0

    def utilization(self) -> float:
        """Коэффициент загрузки сервера (0..1)."""
        return float(np.mean(self.busy)) if self.busy else 0.0
```

```
    def p_idle(self) -> float:
        """Вероятность простоя (P₀ = 1 − ρ)."""
        return 1.0 - self.utilization()


# ПРОЦЕССЫ SimPy

def customer_process(env: simpy.Environment,
                     priority: int,
                     server: simpy.PriorityResource,
                     mu: float,
                     stats: SimStats):
    """
    Жизненный цикл одной заявки:
        1. Запомнить момент прихода.
        2. Встать в очередь с заданным приоритетом (меньший номер = выше приоритет).
        3. Дождаться начала обслуживания (PriorityResource обеспечивает HOL).
        4. Потребить экспоненциально распределённое время обслуживания.
        5. Записать статистику.
    """
    t_arr = env.now
    stats.n_arrived[priority] += 1

    with server.request(priority=priority) as req:
        yield req                           # ← ожидание в очереди
```

```
            t_start = env.now
            svc_time = np.random.exponential(1.0 / mu)
            yield env.timeout(svc_time)                  # ← обслуживание

        t_dep = env.now
        stats.wait_times[priority].append(t_start - t_arr)
        stats.sojourn[priority].append(t_dep - t_arr)
        stats.n_served[priority] += 1


    def source_process(env: simpy.Environment,
                       priority: int,
                       rate: float,
                       server: simpy.PriorityResource,
                       mu: float,
                       stats: SimStats):
        """
        Пуассоновский генератор заявок.
        Интервалы между заявками ~ Exp(1/rate) (свойство пуассоновского процесса).
        """
        while True:
            yield env.timeout(np.random.exponential(1.0 / rate))
            env.process(customer_process(env, priority, server, mu, stats))


    def monitor_process(env: simpy.Environment,
                        server: simpy.PriorityResource,
                        stats: SimStats,
```

```
                delta: float):
    """
    Периодически (каждые delta часов) фиксирует:
      - длину очереди по каждому типу заявок;
      - состояние сервера (занят / свободен).

    Заявки на обслуживание в очереди хранятся как PriorityRequest-объекты
    с атрибутом .priority. Текущее число обслуживаемых заявок: server.count.
    """
    while True:
        stats.ts.append(env.now)
        stats.ql[1].append(sum(1 for r in server.queue if r.priority == 1))
        stats.ql[2].append(sum(1 for r in server.queue if r.priority == 2))
        stats.busy.append(int(server.count > 0))
        yield env.timeout(delta)


# ОСНОВНАЯ ФУНКЦИЯ СИМУЛЯЦИИ
def run_simulation(lambda1: float = LAMBDA1,
                   lambda2: float = LAMBDA2,
                   mu: float      = MU,
                   sim_time: float = SIM_TIME,
                   seed: int       = RANDOM_SEED,
                   delta: float    = MONITOR_DELTA) -> SimStats:
    """
    Запускает дискретно-событийную симуляцию M/M/1 с двумя приоритетами.
```

```
    Аргументы:
      lambda1  — интенсивность потока заявок типа 1 (высокий приоритет)
      lambda2  — интенсивность потока заявок типа 2 (низкий приоритет)
      mu       — интенсивность обслуживания (одинакова для обоих типов)
      sim_time — длительность симуляции (в тех же единицах, что λ и μ)
      seed     — зерно генератора случайных чисел (для воспроизводимости)
      delta    — шаг мониторинга очереди

    Возвращает:
      объект SimStats с накопленной статистикой
    """
    np.random.seed(seed)

    env    = simpy.Environment()
    server = simpy.PriorityResource(env, capacity=1)
    stats  = SimStats()

    # Два независимых генератора заявок
    env.process(source_process(env, 1, lambda1, server, mu, stats))
    env.process(source_process(env, 2, lambda2, server, mu, stats))
    # Процесс мониторинга очереди
    env.process(monitor_process(env, server, stats, delta))
```

```
    env.run(until=sim_time)
    return stats


# АНАЛИТИЧЕСКИЕ ФОРМУЛЫ (M/M/1 HOL, невытесняющий приоритет)

def theory(lambda1: float, lambda2: float, mu: float) -> Optional[dict]:
    """
    Аналитические результаты для модели M/M/1 с HOL (Head-of-Line)
    невытесняющим приоритетом.

    Теоретические формулы (Kleinrock, 1976):

      ρ_k = λ_k / μ  (парциальная загрузка k-го класса)
      ρ   = ρ₁ + ρ₂  (полная загрузка, условие устойчивости: ρ < 1)

    Среднее остаточное время обслуживания (для M/M/1, E[S²] = 2/μ²):
      R₀ = (λ₁ + λ₂) · (2/μ²) / 2 = (λ₁ + λ₂) / μ²

    Среднее время ожидания в очереди:
      W_q(1) = R₀ / (1 − ρ₁)
      W_q(2) = R₀ / [(1 − ρ₁)(1 − ρ₁ − ρ₂)]

    Средняя длина очереди (закон Литтла: L = λ · W):
      L_q(k) = λ_k · W_q(k)

    Вероятность простоя (для M/M/1): P₀ = 1 − ρ
```

```
    Аргументы:
      lambda1, lambda2 — интенсивности поступления
      mu               — интенсивность обслуживания

    Возвращает:
      словарь с аналитическими показателями или None (если система неустойчива)
    """
    rho1 = lambda1 / mu
    rho2 = lambda2 / mu
    rho  = rho1 + rho2

    if rho >= 1.0:
      return None   # система неустойчива, формулы неприменимы

    R0   = (lambda1 + lambda2) / mu**2    # среднее остаточное время обслуживания
    Wq1 = R0 / (1.0 - rho1)
    Wq2 = R0 / ((1.0 - rho1) * (1.0 - rho))
    Lq1 = lambda1 * Wq1                # закон Литтла
    Lq2 = lambda2 * Wq2

    return dict(rho=rho, rho1=rho1, rho2=rho2,
          R0=R0, Wq1=Wq1, Wq2=Wq2,
          Lq1=Lq1, Lq2=Lq2,
          p_idle=1.0 - rho)


# ВЫВОД РЕЗУЛЬТАТОВ В КОНСОЛЬ
```

```
def print_results(stats: SimStats,
                  an: Optional[dict],
                  lambda1: float, lambda2: float, mu: float) -> None:
    """Форматированный вывод сравнительной таблицы в консоль."""

    W = 68
    sep = "─" * W

    print("\n" + "═" * W)
    print("  РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ СМО С ПРИОРИТЕТАМИ")
    print("═" * W)
    print(f"  λ₁ = {lambda1}  λ₂ = {lambda2}  μ = {mu}  "
          f"ρ = {(lambda1+lambda2)/mu:.4f}")
    print(f"  Заявок тип 1: {stats.n_arrived[1]} поступило / "
          f"{stats.n_served[1]} обслужено")
    print(f"  Заявок тип 2: {stats.n_arrived[2]} поступило / "
          f"{stats.n_served[2]} обслужено")
    print()

    # Таблица
    hdr = f"  {'Показатель':<38} {'Тип 1':>10} {'Тип 2':>10}"
    print(sep)
    print(hdr)
    print(sep)

    def row(label, v1, v2, fmt=".3f"):
        print(f"  {label:<38} {v1:>10{fmt}} {v2:>10{fmt}}")
```

```
        # конвертируем часы в минуты
        row("W_q — ср. ожидание (мин, симул.)",
            stats.mean_wait(1) * 60, stats.mean_wait(2) * 60)
        row("L_q — ср. длина очереди (симул.)",
            stats.mean_ql(1), stats.mean_ql(2))

        if an:
            print(sep)
            row("W_q — ср. ожидание (мин, теория)",
                an["Wq1"] * 60, an["Wq2"] * 60)
            row("L_q — ср. длина очереди (теория)",
                an["Lq1"], an["Lq2"])
            print(sep)

            e1 = abs(stats.mean_wait(1) - an["Wq1"]) / an["Wq1"] * 100
            e2 = abs(stats.mean_wait(2) - an["Wq2"]) / an["Wq2"] * 100
            row("Относит. погрешность W_q (%)",  e1, e2, fmt=".2f")

        print(sep)
        print(f"  {'Загрузка сервера ρ (симул.):':<38} {stats.utilization():>10.4f}")
        print(f"  {'Вероятность простоя P₀ (симул.):':<38} {stats.p_idle():>10.4f}")
        if an:
            print(f"  {'Вероятность простоя P₀ (теория):':<38} {an['p_idle']:>10.4f}")
        print("═" * W + "\n")
```

```
# Построение графиков
_BLUE = "#1565C0"
_RED  = "#C62828"
_ORN  = "#E65100"

def _save(fig: plt.Figure, path: str) -> None:
    fig.savefig(path, dpi=150, bbox_inches="tight")
    plt.close(fig)
    print(f"  Сохранён: {path}")



def plot_queue_dynamics(stats: SimStats, out_dir: str,
                        window_hours: float = 100.0) -> None:
    """
    График 1 — динамика длины очереди во времени.
    Показывает первые window_hours часов для читаемости.
    """
    ts = np.array(stats.ts)
    q1 = np.array(stats.ql[1])
    q2 = np.array(stats.ql[2])
    mask = ts <= window_hours

    fig, axes = plt.subplots(2, 1, figsize=(13, 7), sharex=True)
    fig.suptitle("Динамика длины очереди (первые 100 ч моделирования)",
                 fontsize=13, fontweight="bold")

    for ax, data, color, title in zip(
```

```
            axes,
            [q1[mask], q2[mask]],
            [_BLUE, _RED],
            ["Тип 1 — высокий приоритет", "Тип 2 — низкий приоритет"]):
        ax.step(ts[mask], data, color=color, linewidth=0.7, where="post")
        ax.fill_between(ts[mask], data, step="post", alpha=0.25, color=color)
        ax.set_ylabel("Длина очереди", fontsize=10)
        ax.set_title(title, fontsize=10)
        ax.grid(True, alpha=0.3)
        ax.set_ylim(bottom=-0.1)
        mean_val = np.mean(data)
        ax.axhline(mean_val, color=color, linestyle="--",
                   linewidth=1.2, label=f"Среднее = {mean_val:.3f}")
        ax.legend(fontsize=9)

    axes[-1].set_xlabel("Время (часы)", fontsize=10)
    fig.tight_layout()
    _save(fig, os.path.join(out_dir, "queue_dynamics.png"))


def plot_wait_histograms(stats: SimStats, out_dir: str) -> None:
    """График 2 — гистограммы распределения времени ожидания (в минутах)."""
    wt1 = np.array(stats.wait_times[1]) * 60
    wt2 = np.array(stats.wait_times[2]) * 60

    fig, axes = plt.subplots(1, 2, figsize=(13, 5))
    fig.suptitle("Распределение времени ожидания в очереди",
                 fontsize=13, fontweight="bold")
```

```
    for ax, data, color, title in zip(
            axes,
            [wt1, wt2],
            [_BLUE, _RED],
            ["Тип 1 (высокий приоритет)", "Тип 2 (низкий приоритет)"]):
        ax.hist(data, bins=60, color=color, alpha=0.75,
                density=True, edgecolor="white", linewidth=0.4)
        mean_v = np.mean(data)
        ax.axvline(mean_v, color="black", linestyle="--", linewidth=1.8,
                   label=f"Среднее = {mean_v:.2f} мин")
        ax.set_xlabel("Время ожидания (минуты)", fontsize=10)
        ax.set_ylabel("Плотность вероятности", fontsize=10)
        ax.set_title(title, fontsize=11)
        ax.legend(fontsize=9)
        ax.grid(True, alpha=0.3)

    fig.tight_layout()
    _save(fig, os.path.join(out_dir, "wait_histograms.png"))


def plot_boxplot(stats: SimStats, out_dir: str) -> None:
    """График 3 — ящик с усами (box plot) для сравнения типов заявок."""
    wt1 = np.array(stats.wait_times[1]) * 60
    wt2 = np.array(stats.wait_times[2]) * 60

    fig, ax = plt.subplots(figsize=(7, 6))
    bp = ax.boxplot(
```

```
            [wt1, wt2],
            tick_labels=["Тип 1\n(высокий приоритет)", "Тип 2\n(низкий приоритет)"],
            patch_artist=True,
            medianprops=dict(color="white", linewidth=2.5),
            whiskerprops=dict(linewidth=1.4),
            capprops=dict(linewidth=1.4),
            flierprops=dict(marker="o", markersize=2, alpha=0.3),
        )
        colors = [_BLUE, _RED]
        for patch, c in zip(bp["boxes"], colors):
            patch.set_facecolor(c)
            patch.set_alpha(0.75)

        ax.set_ylabel("Время ожидания (минуты)", fontsize=11)
        ax.set_title("Сравнение времени ожидания по типам заявок",
                     fontsize=12, fontweight="bold")
        ax.grid(True, alpha=0.3, axis="y")
        fig.tight_layout()
        _save(fig, os.path.join(out_dir, "wait_boxplot.png"))


    def plot_sim_vs_theory(stats: SimStats, an: dict, out_dir: str) -> None:
        """График 4 — сравнение среднего времени ожидания: симуляция vs теория."""
        wt1_sim = stats.mean_wait(1) * 60
        wt2_sim = stats.mean_wait(2) * 60
        wt1_th  = an["Wq1"] * 60
        wt2_th  = an["Wq2"] * 60
```

```
        categories = ["Тип 1\n(высокий приоритет)", "Тип 2\n(низкий
приоритет)"]
        sim_vals   = [wt1_sim, wt2_sim]
        th_vals    = [wt1_th,  wt2_th]
        x = np.arange(len(categories))
        w = 0.32

        fig, ax = plt.subplots(figsize=(8, 6))
        bars_s = ax.bar(x - w/2, sim_vals, w, label="Симуляция",
                        color=_BLUE, alpha=0.80)
        bars_t = ax.bar(x + w/2, th_vals,  w, label="Аналитика (теория)",
                        color=_ORN, alpha=0.80)

        for bars in (bars_s, bars_t):
            for bar in bars:
                h = bar.get_height()
                ax.annotate(f"{h:.2f}",
                            xy=(bar.get_x() + bar.get_width() / 2, h),
                            xytext=(0, 4), textcoords="offset points",
                            ha="center", va="bottom", fontsize=10)

        ax.set_xticks(x)
        ax.set_xticklabels(categories, fontsize=11)
        ax.set_ylabel("Среднее время ожидания (минуты)", fontsize=11)
        ax.set_title("Симуляция vs. Аналитическая теория",
                     fontsize=12, fontweight="bold")
        ax.legend(fontsize=10)
        ax.grid(True, alpha=0.3, axis="y")
```

```
    fig.tight_layout()
    _save(fig, os.path.join(out_dir, "sim_vs_theory.png"))


def plot_sensitivity(out_dir: str,
                     lambda2: float = LAMBDA2,
                     mu: float      = MU,
                     sim_time: float = 500.0) -> None:
    # График 5 — анализ чувствительности.
    # Показывает, как среднее время ожидания обоих типов изменяется
    # при росте λ₁ (интенсивности высокоприоритетного потока).

    print("  Анализ чувствительности (это займёт ~30 с)...")
    lambda1_grid = np.linspace(0.3, mu - lambda2 - 0.3, 12)
    wq1_sim, wq2_sim = [], []
    wq1_th,  wq2_th  = [], []

    for l1 in lambda1_grid:
        rho = (l1 + lambda2) / mu
        if rho >= 0.99:
            break
        s = run_simulation(lambda1=l1, lambda2=lambda2, mu=mu,
                           sim_time=sim_time, seed=7)
        wq1_sim.append(s.mean_wait(1) * 60)
        wq2_sim.append(s.mean_wait(2) * 60)
        an = theory(l1, lambda2, mu)
        if an:
            wq1_th.append(an["Wq1"] * 60)
            wq2_th.append(an["Wq2"] * 60)
```

```
    n = min(len(wq1_sim), len(wq1_th))
    grid = lambda1_grid[:n]

    fig, ax = plt.subplots(figsize=(9, 5))
    ax.plot(grid, wq1_sim[:n], "o-", color=_BLUE, linewidth=2,
            markersize=5, label="Тип 1 — симуляция")
    ax.plot(grid, wq1_th[:n],  "--", color=_BLUE, linewidth=1.5,
            alpha=0.7, label="Тип 1 — теория")
    ax.plot(grid, wq2_sim[:n], "s-", color=_RED, linewidth=2,
            markersize=5, label="Тип 2 — симуляция")
    ax.plot(grid, wq2_th[:n],  "--", color=_RED, linewidth=1.5,
            alpha=0.7, label="Тип 2 — теория")

    ax.set_xlabel("Интенсивность λ₁ (заявок/ч)", fontsize=11)
    ax.set_ylabel("Среднее время ожидания W_q (мин)", fontsize=11)
    ax.set_title("Влияние λ₁ на время ожидания: симуляция vs. теория",
                 fontsize=12, fontweight="bold")
    ax.legend(fontsize=9)
    ax.grid(True, alpha=0.3)
    fig.tight_layout()
    _save(fig, os.path.join(out_dir, "sensitivity.png"))


# Точка входа
def main():
    rho = (LAMBDA1 + LAMBDA2) / MU
    print("=" * 68)
```

```
    print("  СМО с двумя приоритетами — имитационное моделирование")
    print("=" * 68)
    print(f"  λ₁={LAMBDA1}  λ₂={LAMBDA2}  μ={MU}  "
          f"ρ=(λ₁+λ₂)/μ = {rho:.4f}")
    if rho >= 1.0:
        print("  ОШИБКА: ρ ≥ 1. Система неустойчива. Скорректируйте параметры.")
        return
    print(f"  Время моделирования T = {SIM_TIME} часов")
    print()

    # Основная симуляция
    print("  [1/4] Запуск основной симуляции...")
    stats = run_simulation()

    # Аналитические результаты
    an = theory(LAMBDA1, LAMBDA2, MU)
    print_results(stats, an, LAMBDA1, LAMBDA2, MU)

    # Графики
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    print(f"  [2/4] Построение графиков → директория «{OUTPUT_DIR}/»")
    plot_queue_dynamics(stats, OUTPUT_DIR)
    plot_wait_histograms(stats, OUTPUT_DIR)
    plot_boxplot(stats, OUTPUT_DIR)
    if an:
        plot_sim_vs_theory(stats, an, OUTPUT_DIR)
```

```
    # Анализ чувствительности
    print(f"\n  [3/4] Анализ чувствительности...")
    plot_sensitivity(OUTPUT_DIR)

    print(f"\n  [4/4] Готово! Все графики сохранены в «{OUTPUT_DIR}/»")
    print("=" * 68 + "\n")


if __name__ == "__main__":
    main()
```

Результаты сохраняются в директории results/ в виде файлов PNG. Параметры модели задаются в начале файла (константы LAMBDA1, LAMBDA2, MU, SIM_TIME) без изменения основной логики.

```
# ──────────────────── Параметры ──────────────────────────────────
LAMBDA1      = 2.0    # интенсивность потока тип 1 (высокий приоритет)
LAMBDA2      = 4.0    # интенсивность потока тип 2 (низкий приоритет)
MU           = 8.0    # интенсивность обслуживания
SIM_TIME     = 1000.0 # время моделирования, часов
RANDOM_SEED  = 42     # зерно ГСЧ
MONITOR_DELTA = 0.05  # шаг мониторинга очереди, часов
OUTPUT_DIR   = 'results'
```

```
# ──────────────────── Точка входа
────────────────────────────────────────

if __name__ == '__main__':
    stats = run_simulation()
    an    = theory(LAMBDA1, LAMBDA2, MU)
    print_results(stats, an, LAMBDA1, LAMBDA2, MU)
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    plot_queue_dynamics(stats, OUTPUT_DIR)
    plot_wait_histograms(stats, OUTPUT_DIR)
    plot_boxplot(stats, OUTPUT_DIR)
    plot_sim_vs_theory(stats, an, OUTPUT_DIR)
    plot_sensitivity(OUTPUT_DIR)
```