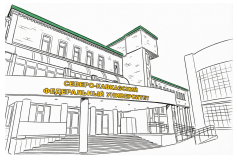

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

НАДЁЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания
к выполнению лабораторных работ

Направление подготовки: 09.03.02 «Информационные системы и технологии»
Профиль: «Прикладное программирование и интернет-технологии»

Квалификация (степень) выпускника: бакалавр
Форма обучения: очная

Ставрополь – 2026

УДК 004.056:519.876
ББК 32.973.202-018.2

НАДЁЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания к лабораторным работам

Аннотация: Методические указания содержат комплект из 5 лабораторных работ по дисциплине «Надёжность информационных систем». Каждая работа включает цель, задачи, теоретический минимум, пошаговый разбор примера, фрагменты программного кода для расчёта показателей надёжности и моделирования отказов, индивидуальное задание для самостоятельного выполнения, а также вопросы для самоконтроля. Материал охватывает ключевые методы количественной оценки, структурного анализа и практического обеспечения отказоустойчивости программно-аппаратных комплексов. Рекомендуемый объем — 20 часов лабораторных занятий. Работы выполняются с использованием среды Python и специализированных библиотек (SciPy, NumPy, matplotlib) для аналитических расчётов и имитационного моделирования.
Составитель: к.т.н. Д.Л. Винокурский
Рецензент: к.ф.м.н. Е.В. Крахоткина

Содержание

Введение

Дисциплина «Надёжность информационных систем» является критически важной для подготовки специалистов, способных проектировать и эксплуатировать системы, функционирующие в условиях неопределённости, внешних возмущений и внутренних сбоев. Понимание физических и математических основ надёжности, умение количественно оценивать риски и применять методы повышения отказоустойчивости — необходимые компетенции для разработки современных облачных сервисов, критических инфраструктур и систем реального времени.

Актуальность дисциплины обусловлена следующими факторами:

- Рост зависимости бизнеса и общества от непрерывной работы ИТ-сервисов повышает требования к доступности (99.99% и выше);
- Усложнение архитектур (микросервисы, распределённые системы) увеличивает количество потенциальных точек отказа;
- Киберугрозы и аппаратные сбои требуют превентивных методов анализа и прогнозирования отказов;
- Регуляторные требования (ФЗ-152, ГОСТ Р МЭК 61508) предписывают документирование показателей надёжности.

Цель методических указаний — предоставить обучающимся структурированный практический материал для освоения методов расчёта, анализа и обеспечения надёжности информационных систем.

Задачи лабораторного практикума:

1. Сформировать навыки расчёта базовых показателей надёжности (наработка на отказ, вероятность безотказной работы, коэффициент готовности);
2. Научить выполнять структурный анализ надёжности сложных систем с использованием аналитических методов;
3. Освоить методики построения и анализа деревьев отказов (FTA) и таблиц FMEA;
4. Развить способность применять паттерны отказоустойчивости при проектировании программных сервисов;
5. Подготовить к решению прикладных задач оценки рисков и выбора стратегий резервирования.

Организация выполнения работ:

- Каждая лабораторная работа рассчитана на 4 академических часа;
- Перед началом работы студент обязан изучить теоретический материал и ответить на допусковые вопросы;
- Экспериментальная часть выполняется в среде Python с использованием синтетических данных и моделей отказов;
- Отчёт по работе должен содержать: цель, теоретические формулы, листинги кода, таблицы результатов, графики, выводы и ответы на контрольные вопросы;
- Защита работы включает демонстрацию работоспособности скриптов и устный ответ по методологии анализа надёжности.

Требования к программному обеспечению:

- Язык программирования: Python 3.9+;
- Библиотеки: NumPy, SciPy (stats, optimize), Matplotlib, Pandas, NetworkX (для FTA);
- Среда разработки: Jupyter Notebook, VS Code или PyCharm;

- Дополнительно: Graphviz (для визуализации деревьев отказов), Docker (для тестирования паттернов устойчивости).

1 Расчёт базовых показателей надёжности: наработка на отказ, вероятность безотказной работы, доступность

Цель: Освоить методы количественной оценки надёжности невосстанавливаемых и восстанавливаемых систем на основе статистических данных об отказах.

Задачи:

- Изучить основные показатели надёжности: $MTBF$, $MTTR$, $P(t)$, $\lambda(t)$, K_g;
- Освоить методы оценки параметров экспоненциального и Вейбулла-распределений по выборке данных;
- Научиться рассчитывать вероятность безотказной работы и коэффициент готовности для заданных условий эксплуатации;
- Реализовать программную обработку экспериментальных данных и визуализацию функций надёжности.

Теоретическое описание: Для невосстанавливаемых элементов вероятность безотказной работы определяется как $P(t) = \Pr\{T > t\}$, где T — случайное время до отказа. При экспоненциальном законе распределения:

$$P(t) = e^{-\lambda t}, \quad f(t) = \lambda e^{-\lambda t}, \quad MTBF = \frac{1}{\lambda}, \tag{1}$$

где λ — интенсивность отказов (постоянная). Для восстанавливаемых систем вводится коэффициент готовности:

$$K_g = \frac{MTBF}{MTBF + MTTR}, \tag{2}$$

где $MTTR$ — среднее время восстановления. Распределение Вейбулла обобщает экспоненциальное:

$$P(t) = \exp\left[-\left(\frac{t}{\eta}\right)^{\beta}\right], \tag{3}$$

где β — параметр формы (характеризует тип отказов), η — масштабный параметр.

Разобранный пример: 1. Выборка: времена наработки до отказа 20 серверов (часы): $[1250, 3420, \ldots, 2890]$.
2. Оценка λ методом максимального правдоподобия: $\hat{\lambda} = n / \sum t_i$.
3. Расчёт $P(1000\,\text{ч}) = e^{-\hat{\lambda}\cdot 1000}$ и K_g при $MTTR = 4$ ч.
4. Результат: $\hat{\lambda} = 2.1 \cdot 10^{-4}$ ч$^{-1}$, $P(1000) \approx 0.81$, $K_g \approx 0.988$.

Программный код (оценка параметров и расчёт показателей):

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

#                                                    :
                  (         )
t_data = np.array([1250, 3420, 890, 2100, 4560, 1780, 3200, 950,
                   2890, 1560, 3800, 2200, 1100, 4100, 2650,
                   1890, 3300, 2450, 1670, 2980])

#
                                (MLE)
lambda_hat = len(t_data) / np.sum(t_data)
```

```
mtbf_hat = 1 / lambda_hat
print(f"                                                    :     =
    {lambda_hat:.3e}         ")
print(f"                                                 : MTBF = {
    mtbf_hat:.1f}    ")

#                                                                t
     = 1000
t_target = 1000
P_t = np.exp(-lambda_hat * t_target)
print(f"P({t_target}    ) = {P_t:.3f}")

#                                                    MTTR = 4
mttr = 4.0
K_g = mtbf_hat / (mtbf_hat + mttr)
print(f"                                             : K_    = {K_g:.4f}
    ({K_g*100:.2f}%)")

#
t_plot = np.linspace(0, 5000, 200)
P_plot = np.exp(-lambda_hat * t_plot)

plt.figure(figsize=(8, 5))
plt.plot(t_plot, P_plot, 'b-', linewidth=2, label='P(t) = exp(- t )')
plt.axvline(t_target, color='r', linestyle='--', label=f't = {
    t_target}    ')
plt.axhline(P_t, color='r', linestyle=':', alpha=0.7)
plt.xlabel('           ,    '); plt.ylabel('
                                    ')
plt.title('
                                                 ');
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout()
plt.show()
```

Индивидуальное задание: По выборке из 30 значений времени до отказа оцените параметры распределения Вейбулла (β, η) методом максимального правдоподобия (используйте `scipy.stats.weibull_min.fit`). Постройте график $P(t)$ для экспоненциальной и Вейбулла-моделей, сравните прогнозы на горизонте 2000 ч.

Вопросы для самопроверки:

1. При каких условиях экспоненциальная модель адекватно описывает процесс отказов?
2. Как параметр формы β распределения Вейбулла интерпретируется с физической точки зрения?
3. Почему коэффициент готовности не зависит от времени для стационарного процесса восстановления?

2 Структурный анализ надёжности: последовательные, параллельные и мостиковые схемы

Цель: Научиться строить расчётные схемы надёжности и выполнять количественную оценку систем со сложной структурой связей.

Задачи:

- Изучить методы расчёта надёжности последовательных, параллельных и смешанных структур;
- Освоить технику свёртки структур и метод разложения относительно элемента;
- Реализовать алгоритм расчёта вероятности безотказной работы для мостиковой схемы;
- Проанализировать влияние резервирования на общую надёжность системы.

Теоретическое описание: Для последовательного соединения n независимых элементов:

$$P_{sys}(t) = \prod_{i=1}^{n} P_i(t). \tag{4}$$

Для параллельного (нагруженного) резервирования:

$$P_{sys}(t) = 1 - \prod_{i=1}^{n} [1 - P_i(t)]. \tag{5}$$

Мостиковая схема (5 элементов) не сводится к простым последовательно-параллельным комбинациям. Её надёжность рассчитывается методом разложения относительно ключевого элемента k:

$$P_{sys} = P_k \cdot P_{sys|k=1} + (1 - P_k) \cdot P_{sys|k=0}, \tag{6}$$

где $P_{sys|k=1}$ и $P_{sys|k=0}$ — надёжности системы при исправном и отказавшем элементе k соответственно.

Разобранный пример: 1. Система: 5 одинаковых элементов с $P = 0.95$, соединённых мостиком.
2. Разложение относительно центрального элемента: при его исправности схема сводится к двум параллельным цепям, при отказе — к последовательному соединению двух параллельных пар.
3. Расчёт: $P_{sys} = 0.95 \cdot [1 - (1 - 0.95^2)^2] + 0.05 \cdot [1 - (1 - 0.95)^2]^2 \approx 0.994$.
4. Вывод: мостиковая структура обеспечивает надёжность выше, чем простое параллельное резервирование трёх элементов (0.999875 против 0.999), но с меньшими затратами.

Программный код (расчёт надёжности мостиковой схемы):

```
import numpy as np
import matplotlib.pyplot as plt

def bridge_reliability(P):
    """
                          5                                                """
    #

    P_center = P
```

```
    #                                                    :
                                                (                    1-3
       2-4)
    P_chain = 1 - (1 - P**2)**2   #

    #                                                      :


    P_pair = 1 - (1 - P)**2   #
    P_series_pairs = P_pair**2   #

    #
    return P_center * P_chain + (1 - P_center) * P_series_pairs

#
P_elem = np.linspace(0.5, 0.99, 50)
P_series = P_elem**5   # 5
P_parallel = 1 - (1 - P_elem)**5   # 5
P_bridge = [bridge_reliability(p) for p in P_elem]

plt.figure(figsize=(9, 6))
plt.plot(P_elem, P_series, 'r--', label='
                                    (5      .)')
plt.plot(P_elem, P_parallel, 'g-.', label='
   (5      .)')
plt.plot(P_elem, P_bridge, 'b-', linewidth=2, label='
                          (5      .)')
plt.xlabel('                                          $P$'); plt.ylabel('
                                      $P_{sys}$')
plt.title('
                         ');
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout()
plt.show()

#                                        P = 0.95
P_test = 0.95
print(f"                                                {P_test}:")
print(f"                                     : {P_test**5:.4f}")
print(f"                                : {1 - (1-P_test)**5:.4f}")
print(f"                           : {bridge_reliability(P_test):.4f}")
```

Индивидуальное задание: Реализуйте функцию расчёта надёжности для произвольной структуры, заданной матрицей смежности графа и вектором надёжностей элементов (используйте метод перебора состояний или алгоритм факторизации). Протестируйте на схеме «2 из 3» и сравните результаты с аналитическим расчётом.

Вопросы для самопроверки:

1. Почему надёжность последовательной системы всегда ниже надёжности наименее надёжного элемента?
2. В чём преимущество нагруженного резервирования перед резервированием замещением?
3. Как метод разложения относительно элемента упрощает расчёт сложных структур?

3 Построение и анализ деревьев отказов (Fault Tree Analysis, FTA)

Цель: Освоить дедуктивный метод анализа причин критических отказов с использованием логических моделей деревьев отказов.

Задачи:

- Изучить базовые элементы FTA: вершинное событие, промежуточные события, базовые отказы, логические вентили (И, ИЛИ);
- Научиться строить дерево отказов для типовой ИС (например, отказ сервиса веб-приложения);
- Реализовать расчёт вероятности вершинного события на основе вероятностей базовых отказов;
- Выполнить анализ минимальных сечений (minimal cut sets) для выявления критических комбинаций отказов.

Теоретическое описание: Дерево отказов — направленная ациклическая графовая модель, связывающая вершинное событие (нежелательное состояние системы) с комбинациями базовых событий (отказы компонентов) через логические вентили. Для вентиля **ИЛИ** вероятность выходного события:

$$P_{out} = 1 - \prod_{i=1}^{n}(1 - P_i), \tag{7}$$

для вентиля **И**:

$$P_{out} = \prod_{i=1}^{n} P_i. \tag{8}$$

Минимальное сечение — наименьшая комбинация базовых событий, приводящая к вершинному отказу. Вероятность вершинного события приближённо оценивается суммой вероятностей минимальных сечений (при малых P_i).

Разобранный пример: 1. Вершинное событие: «Недоступность веб-сервиса».
2. Базовые события: отказ сервера ($P_1 = 0.01$), сбой БД ($P_2 = 0.02$), обрыв сети ($P_3 = 0.005$), ошибка в коде ($P_4 = 0.015$).
3. Логика: отказ сервиса = (отказ сервера ИЛИ сбой БД) И (обрыв сети ИЛИ ошибка в коде).
4. Расчёт: $P_{top} = [1 - (1 - 0.01)(1 - 0.02)] \cdot [1 - (1 - 0.005)(1 - 0.015)] \approx 0.00049$.

Программный код (моделирование дерева отказов):

```
import numpy as np
from dataclasses import dataclass
from typing import List, Union

@dataclass
class Event:
    name: str
    probability: float  #
    children: List['Event'] = None  #

    gate: str = None  # 'AND'        'OR'

    def is_basic(self):
```

```
        return self.children is None

    def calculate_probability(self):
        if self.is_basic():
            return self.probability
        child_probs = [child.calculate_probability() for child in
           self.children]
        if self.gate == 'OR':
            # P = 1 - prod(1 - P_i)
            return 1 - np.prod([1 - p for p in child_probs])
        elif self.gate == 'AND':
            # P = prod(P_i)
            return np.prod(child_probs)
        else:
            raise ValueError(f"                                    :
                {self.gate}")

#
#
server_fail = Event("                          ", 0.01)
db_fail = Event("             ", 0.02)
network_fail = Event("                     ", 0.005)
code_error = Event("                        ", 0.015)

#
infra_fail = Event("                                              ",
   None,
                   children=[server_fail, db_fail], gate='OR')
logic_fail = Event("                                  ", None,
                   children=[network_fail, code_error], gate='OR')

#
top_event = Event("                                             ", None,
                  children=[infra_fail, logic_fail], gate='AND')

#
P_top = top_event.calculate_probability()
print(f"                                                    : {
   P_top:.5f} ({P_top*100:.3f}%)")

#                                                  (
                    :
                   )
def find_minimal_cut_sets(event, path=None):
    """
                    """
    if path is None:
        path = []
    if event.is_basic():
        return [[event.name]]

    if event.gate == 'OR':
        #              :
```

```
        cuts = []
        for child in event.children:
            cuts.extend(find_minimal_cut_sets(child, path))
        return cuts
    elif event.gate == 'AND':
        #           :

        child_cuts = [find_minimal_cut_sets(c) for c in event.
           children]
        result = []
        def combine(lists, idx=0, current=[]):
            if idx == len(lists):
                result.append(current[:])
                return
            for cut in lists[idx]:
                combine(lists, idx+1, current + cut)
        combine(child_cuts)
        return result
    return []

cuts = find_minimal_cut_sets(top_event)
print(f"\ n                                    ({len(cuts)}     .):"
   )
for i, cut in enumerate(cuts, 1):
    print(f"  {i}. {'    '.join(cut)}")
```

Индивидуальное задание: Постройте дерево отказов для системы резервного копирования данных. Включите базовые события: сбой диска, ошибка ПО, человеческий фактор, внешние воздействия. Рассчитайте вероятность потери данных за год при заданных вероятностях базовых отказов. Определите, какие минимальные сечения вносят наибольший вклад в риск.

Вопросы для самопроверки:

1. В чём принципиальное отличие дедуктивного (FTA) и индуктивного (FMEA) подходов к анализу отказов?
2. Почему при расчёте вероятности вершинного события через минимальные сечения используется приближение суммы?
3. Как учитывать зависимые отказы в рамках классического FTA?

4 Анализ видов и последствий отказов (FMEA) для программного компонента

Цель: Сформировать навыки превентивной оценки рисков на уровне проектных решений с использованием методики FMEA.

Задачи:

- Изучить структуру таблицы FMEA: режим отказа, причина, следствие, критичность (S, O, D, RPN);
- Научиться идентифицировать потенциальные отказы программного модуля на этапе проектирования;
- Освоить методику расчёта приоритетного числа риска (RPN) и ранжирования угроз;
- Разработать корректирующие действия для снижения критических рисков.

Теоретическое описание: FMEA (Failure Mode and Effects Analysis) — индуктивный метод анализа, направленный на выявление потенциальных отказов и их последствий. Для каждого режима отказа оцениваются три параметра по шкале 1–10:

- **Severity (S)** — тяжесть последствий для системы/пользователя;
- **Occurrence (O)** — вероятность возникновения причины отказа;
- **Detection (D)** — вероятность обнаружения отказа до наступления последствий.

Приоритетное число риска: $RPN = S \times O \times D$. Мероприятия планируются для элементов с $RPN > 100$ или $S \geq 9$.

Разобранный пример: 1. Объект: модуль аутентификации веб-приложения.
2. Выявленные режимы отказов: 1) утечка токенов, 2) блокировка легитимных пользователей, 3) обход аутентификации.
3. Оценка для «утечка токенов»: $S = 9$ (компрометация данных), $O = 4$ (редко при правильном хранении), $D = 6$ (логирование есть, но не в реальном времени) $\rightarrow RPN = 216$.
4. Корректирующее действие: внедрение короткоживущих токенов + refresh-механизм $\rightarrow$ снижение O до 2, $RPN = 108$.

Программный код (автоматизация расчёта RPN и генерации отчёта):

```
import pandas as pd
import numpy as np

#                                    FMEA
class FMEAEntry:
    def __init__(self, component, failure_mode, cause, effect, S, O,
       D, action=None):
        self.component = component
        self.failure_mode = failure_mode
        self.cause = cause
        self.effect = effect
        self.S = S  # Severity 1-10
        self.O = O  # Occurrence 1-10
        self.D = D  # Detection 1-10
        self.RPN = S * O * D
        self.action = action
        self.RPN_after = None
        if action:
            #              :                                    O     D
```

```
                    2             (               1)
            self.O_after = max(1, O - 2)
            self.D_after = max(1, D - 2)
            self.RPN_after = S * self.O_after * self.D_after

    def to_dict(self):
        return {
            '                  ': self.component,
            '                       ': self.failure_mode,
            '               ': self.cause,
            '                        ': self.effect,
            'S': self.S, 'O': self.O, 'D': self.D, 'RPN': self.RPN,
            '                  ': self.action or '-',
            'RPN            ': self.RPN_after if self.RPN_after else '
               -'
        }

#              : FMEA

entries = [
    FMEAEntry("                              ", "
                          ",
            "                                ", "
                                             ", 9, 3, 7,
            "                          idempotency key + retry
                                                                  "
                ),
    FMEAEntry("                                    ", "
                                             ",
            "
                                    ", "
                                             ", 8, 4, 5,
            "                                -
                                                  +
                                                     "),
    FMEAEntry("       -         ", "
                  ",
            "                                                                ", "
                                                                               ", 6,
                5, 6,
            "                    TTL +
                                         "),
    FMEAEntry("              ", "
                  ",
            "                                          ", "

                                                                  ", 7,
                2, 8,
            "                                                    +

                                                          "),
]

```

```
#                                DataFrame                                RPN
df = pd.DataFrame([e.to_dict() for e in entries])
df = df.sort_values('RPN', ascending=False).reset_index(drop=True)

#
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 120)
print("                FMEA (
                    RPN):\n")
print(df.to_string(index=False))

#
critical = df[df['RPN'] >= 100]
print(f"\ n                                    (RPN     100): {len(
   critical)}      {len(df)}")
if not critical.empty:
    print("                                                  :")
    for _, row in critical.iterrows():
        print(f"      {row['                          ']}: RPN={row['RPN
           ']}     {row['                ']}")
```

Индивидуальное задание: Выполните FMEA-анализ для модуля обработки файлов в облачном хранилище. Выявите не менее 6 режимов отказов, оцените S/O/D, рассчитайте RPN. Для двух наиболее критичных рисков предложите конкретные технические меры снижения и пересчитайте RPN после внедрения.

Вопросы для самопроверки:

1. Почему параметр Detection (D) в FMEA часто недооценивают, и к каким последствиям это приводит?
2. В каких случаях целесообразно использовать модификацию FMECA (с учётом критичности)?
3. Как интегрировать результаты FMEA в процесс непрерывной интеграции (CI/CD)?

5 Проектирование отказоустойчивого сервиса с применением паттернов устойчивости

Цель: Реализовать практические механизмы повышения живучести распределённого приложения с использованием паттернов отказоустойчивости.

Задачи:

- Изучить паттерны: Circuit Breaker, Retry, Bulkhead, Timeout, Fallback;
- Реализовать на Python декораторы/классы для внедрения паттернов в существующий код;
- Протестировать поведение сервиса при моделировании сбоев внешних зависимостей;
- Оценить влияние паттернов на метрики доступности и времени отклика.

Теоретическое описание: Circuit Breaker предотвращает каскадные отказы, размыкая цепь вызова после N последовательных ошибок и периодически пробуя восстановить соединение. Состояния: Closed (нормальная работа) → Open (блокировка вызовов) → Half-Open (пробный вызов).
Retry с экспоненциальной задержкой ($delay = base \cdot 2^{attempt}$) компенсирует временные сбои, но требует идемпотентности операций.
Bulkhead изолирует ресурсы для разных типов нагрузки, предотвращая истощение пула потоков/соединений.
Timeout ограничивает время ожидания ответа, освобождая ресурсы при зависании внешних сервисов.

Разобранный пример: 1. Сервис: клиент API погоды с внешним вызовом `fetch_weather()`.
2. Внедрение: декоратор `@circuit_breaker` с параметрами `failure_threshold=5`, `recovery_timeout=30`.
3. Тест: моделирование 7 последовательных сбоев → переход в состояние Open, блокировка вызовов на 30 с.
4. Результат: при восстановлении внешнего сервиса автоматический возврат в Closed после успешного пробного вызова.

Программный код (реализация Circuit Breaker и Retry):

```
import time
import random
from functools import wraps
from enum import Enum
from typing import Callable, Optional

class CircuitState(Enum):
    CLOSED = "closed"
    OPEN = "open"
    HALF_OPEN = "half_open"

class CircuitBreaker:
    def __init__(self, failure_threshold: int = 5,
                 recovery_timeout: float = 30.0,
                 expected_exception: type = Exception):
        self.failure_threshold = failure_threshold
        self.recovery_timeout = recovery_timeout
```

```
        self.expected_exception = expected_exception
        self.failure_count = 0
        self.last_failure_time: Optional[float] = None
        self.state = CircuitState.CLOSED

    def call(self, func: Callable, *args, **kwargs):
        if self.state == CircuitState.OPEN:
            if time.time() - self.last_failure_time > self.
               recovery_timeout:
                self.state = CircuitState.HALF_OPEN
                print("                    HALF_OPEN:
                                          ")
            else:
                raise RuntimeError("Circuit is OPEN     call blocked"
                   )

        try:
            result = func(*args, **kwargs)
            self._on_success()
            return result
        except self.expected_exception as e:
            self._on_failure()
            raise e

    def _on_success(self):
        self.failure_count = 0
        self.state = CircuitState.CLOSED

    def _on_failure(self):
        self.failure_count += 1
        self.last_failure_time = time.time()
        if self.failure_count >= self.failure_threshold:
            self.state = CircuitState.OPEN
            print(f"    Circuit OPEN           {self.failure_count}
                            ")

    def __call__(self, func: Callable) -> Callable:
        @wraps(func)
        def wrapper(*args, **kwargs):
            return self.call(func, *args, **kwargs)
        return wrapper

def retry_with_backoff(max_attempts: int = 3,
                       base_delay: float = 1.0,
                       exceptions: tuple = (Exception,)):
    def decorator(func: Callable) -> Callable:
        @wraps(func)
        def wrapper(*args, **kwargs):
            last_exc = None
            for attempt in range(max_attempts):
                try:
                    return func(*args, **kwargs)
                except exceptions as e:
```

```
                    last_exc = e
                    if attempt == max_attempts - 1:
                        raise
                    delay = base_delay * (2 ** attempt)  #

                    print(f"                    {attempt+1}
                                        ,                                {delay
                       :.1f}   ")
                    time.sleep(delay)
            raise last_exc
        return wrapper
    return decorator

#
@CircuitBreaker(failure_threshold=3, recovery_timeout=10)
@retry_with_backoff(max_attempts=2, base_delay=0.5)
def fetch_weather(city: str) -> dict:
    """                                                API
                                        40%"""
    if random.random() < 0.4:
        raise ConnectionError(f"Timeout connecting to weather API for
            {city}")
    return {"city": city, "temp": random.randint(-10, 35), "status":
       "OK"}

#
print("
                  :\n")
for i in range(10):
    try:
        result = fetch_weather("Moscow")
        print(f"{i+1}.           : {result}")
    except Exception as e:
        print(f"{i+1}.             : {type(e).__name__}: {e}")
    time.sleep(1)  #
```

Индивидуальное задание: Расширьте пример, добавив паттерны Bulkhead (ограничение параллельных вызовов через `asyncio.Semaphore`) и Fallback (возврат кэшированных данных при недоступности основного источника). Проведите нагрузочное тестирование: сравните процент успешных запросов и среднее время отклика для базовой и отказоустойчивой версий при 20% сбоев внешнего API.

Вопросы для самопроверки:

1. Почему идемпотентность является обязательным условием для безопасного применения Retry?
2. Как выбрать параметры `failure_threshold` и `recovery_timeout` для Circuit Breaker в реальных условиях?
3. В чём компромисс между строгой консистентностью и доступностью при использовании Fallback-механизмов?

Список литературы

[1] Гнеденко, Б. В. Математические методы в теории надёжности / Б. В. Гнеденко, Ю. К. Беляев, А. Д. Соловьёв. — 3-е изд., перераб. и доп. — Москва : Издательство ЛКИ, 2021. — 584 с. — ISBN 978-5-382-02145-3.

[2] Липаев, В. В. Надёжность программных средств информационных систем : монография / В. В. Липаев. — Москва : Издательство Юрайт, 2023. — 421 с. — (Научная мысль). — ISBN 978-5-534-16892-4. — URL: https://urait.ru/bcode/521456 (дата обращения: 01.06.2026). — Текст : электронный.

[3] Григорьев, В. В. Надежность информационных систем : учебное пособие для вузов / В. В. Григорьев, А. С. Мартынов. — Москва : Издательство Юрайт, 2023. — 267 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-15633-9. — URL: https://urait.ru/bcode/518745 (дата обращения: 01.06.2026). — Текст : электронный.

[4] Клеандрова, М. И. Теория надежности и риски : учебное пособие / М. И. Клеандрова. — Санкт-Петербург : Издательство «Лань», 2021. — 196 с. — (Учебники для вузов. Специальная литература). — ISBN 978-5-8114-8234-1. — URL: https://e.lanbook.com/book/256789 (дата обращения: 01.06.2026). — Текст : электронный.

[5] Avizienis A., Laprie J.-C., Randell B., Landwehr C. Basic concepts and taxonomy of dependable and secure computing // IEEE Transactions on Dependable and Secure Computing. — 2004. — Vol. 1, № 1. — P. 11–33. — DOI: 10.1109/TDSC.2004.1.

[6] Nyberg M. Risk Analysis and Fault Tree Analysis: A Practical Guide / M. Nyberg, L. Blanke. — Springer, 2022. — 312 p. — ISBN 978-3-030-98765-4.

[7] Richardson C. Microservices Patterns: With examples in Java / C. Richardson. — Manning Publications, 2018. — 520 p. — ISBN 978-1-61729-454-9.

[8] Google. Site Reliability Engineering: How Google Runs Production Systems / B. Beyer et al. — O'Reilly Media, 2016. — 552 p. — ISBN 978-1-4919-2912-4.

[9] Virtanen P. et al. SciPy 1.0: fundamental algorithms for scientific computing in Python // Nature Methods. — 2020. — Vol. 17. — P. 261–272. — DOI: 10.1038/s41592-019-0686-2.

[10] The pandas development team. pandas-dev/pandas: Pandas. — Zenodo, 2024. — DOI: 10.5281/zenodo.3509134.