Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №8
дисциплины «Объектно-ориентированное программирование»
Вариант 7

Выполнил:
Мотовилов Вадим Борисович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А. ктн доцент департамента перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2026 г.

Тема: синхронизация потоков в языке Python.

Цель работы: приобретение навыков использования примитивов синхронизации при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Задание 1.

С использованием многопоточности для заданного значения x необходимо организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результаты вычисления должны передаваться второй функции, вычисляемой в отдельном потоке. Потоки для вычисления значений двух функций должны запускаться одновременно. Значение сумм рядов должны быть вычислены с точностью $\varepsilon = 10^{-7}$. Номера вариантов необходимо уточнить у преподавателя:

$$S = \sum_{n=1}^{\infty} \frac{(-1)^{n+1} \sin nx}{n} = \sin x - \frac{\sin 2x}{2} + \ldots;$$

$$x = -\frac{\pi}{2}; y = \frac{x}{2}.$$

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math
import threading
import time
import queue
from dataclasses import dataclass
from typing import Optional, Tuple
import matplotlib.pyplot as plt

@dataclass
class SeriesResult:
    """Результат вычисления суммы ряда"""

    value: float
    iterations: int
    calculation_time: float
    thread_id: int

class SeriesCalculator:
    """Класс для вычисления суммы ряда"""

    def __init__(self, x: float, epsilon: float = 1e-7):
        self.x = x
        self.epsilon = epsilon
```

```
        self.stop_event = threading.Event()

    def term(self, n: int) -> float:
        """Вычисление n-го члена ряда"""
        return ((-1) ** (n + 1) * math.sin(n * self.x)) / n

    def calculate_sum(
        self,
        start_n: int = 1,
        result_queue: Optional[queue.Queue] = None,
        thread_id: int = 0,
    ) -> SeriesResult:
        """Вычисление суммы ряда (функция для потока)"""
        start_time = time.time()
        sum_value = 0.0
        n = start_n
        term_value = self.term(n)

        while abs(term_value) >= self.epsilon and not self.stop_event.is_set():
            sum_value += term_value
            n += 1
            term_value = self.term(n)

        # Если достигли точности, сигнализируем другим потокам
        if abs(term_value) < self.epsilon:
            self.stop_event.set()

        end_time = time.time()
        result = SeriesResult(
            value=sum_value,
            iterations=n - start_n,
            calculation_time=end_time - start_time,
            thread_id=thread_id,
        )

        if result_queue is not None:
            result_queue.put(result)

        return result

class PipelineStage:
    """Стадия конвейерной обработки"""

    def __init__(self, name: str):
        self.name = name
        self.input_queue = queue.Queue()
        self.output_queue = queue.Queue()
        self.thread = None
        self.results = []

    def start(self):
        """Запуск стадии конвейера"""
```

```
        self.thread = threading.Thread(target=self._process)
        self.thread.start()

    def _process(self):
        """Обработка данных (должен быть переопределен в наследниках)"""
        pass

    def wait(self):
        """Ожидание завершения стадии"""
        if self.thread:
            self.thread.join()

class FirstFunctionStage(PipelineStage):
    """Первая стадия конвейера: вычисление суммы ряда"""

    def __init__(self, x: float, epsilon: float = 1e-7):
        super().__init__("Первая функция (ряд)")
        self.calculator = SeriesCalculator(x, epsilon)

    def _process(self):
        """Вычисление суммы ряда"""
        try:
            result = self.calculator.calculate_sum(
                result_queue=self.output_queue, thread_id=1
            )
            self.results.append(result)
            print(
                f"[{self.name}] Вычисление завершено: "
                f"S = {result.value:.10f}, "
                f"итераций: {result.iterations}, "
                f"время: {result.calculation_time:.6f} с"
            )
        except Exception as e:
            print(f"[{self.name}] Ошибка: {e}")
            self.output_queue.put(None)

class SecondFunctionStage(PipelineStage):
    """Вторая стадия конвейера: обработка результатов первой функции"""

    def __init__(self):
        super().__init__("Вторая функция (обработка)")
        self.control_value = None

    def set_control_value(self, y: float):
        """Установка контрольного значения"""
        self.control_value = y

    def _process(self):
        """Обработка результатов первой функции"""
        try:
            # Ждем результат от первой функции
            input_data = self.input_queue.get(timeout=5.0)
```

```
            if input_data is None:
                print(f"[{self.name}] Получен пустой результат")
                self.output_queue.put(None)
                return

            # Вычисляем контрольное значение
            if self.control_value is None:
                print(f"[{self.name}] Ошибка: контрольное значение не установлено")
                self.output_queue.put(None)
                return

            # Выполняем обработку: вычисляем ошибки
            start_time = time.time()
            time.sleep(0.01)  # Имитация вычислений

            absolute_error = abs(input_data.value - self.control_value)
            relative_error = (
                absolute_error / abs(self.control_value)
                if self.control_value != 0
                else float("inf")
            )

            end_time = time.time()

            result = {
                "series_value": input_data.value,
                "control_value": self.control_value,
                "absolute_error": absolute_error,
                "relative_error": relative_error,
                "iterations": input_data.iterations,
                "processing_time": end_time - start_time,
                "total_time": input_data.calculation_time + (end_time - start_time),
            }

            self.results.append(result)
            print(
                f"[{self.name}] Обработка завершена: "
                f"абс. ошибка = {absolute_error:.2e}, "
                f"отн. ошибка = {relative_error:.2e}"
            )

            self.output_queue.put(result)

        except queue.Empty:
            print(f"[{self.name}] Таймаут ожидания данных")
            self.output_queue.put(None)
        except Exception as e:
            print(f"[{self.name}] Ошибка: {e}")
            self.output_queue.put(None)

class Pipeline:
```

```
"""Конвейер для обработки данных"""

def __init__(self, x: float, epsilon: float = 1e-7):
    self.x = x
    self.epsilon = epsilon
    self.stage1 = FirstFunctionStage(x, epsilon)
    self.stage2 = SecondFunctionStage()

    # Соединяем стадии: выход stage1 -> вход stage2
    self.stage2.input_queue = self.stage1.output_queue

def run(self) -> Tuple[Optional[dict], Optional[SeriesResult]]:
    """Запуск конвейера"""
    print("\n" + "=" * 70)
    print("Запуск конвейерной обработки")
    print("=" * 70)

    # Вычисляем контрольное значение
    y = self.x / 2
    self.stage2.set_control_value(y)

    print(f"Параметры:")
    print(f"  x = {self.x:.6f}")
    print(f"  Контрольное значение y = x/2 = {y:.10f}")
    print(f"  Точность ε = {self.epsilon:.2e}")
    print()

    # Запускаем обе стадии одновременно
    print("[Конвейер] Запуск стадий...")
    start_time = time.time()

    self.stage1.start()
    self.stage2.start()

    # Ждем завершения обеих стадий
    print("[Конвейер] Ожидание завершения вычислений...")

    self.stage1.wait()
    self.stage2.wait()

    end_time = time.time()
    total_time = end_time - start_time

    # Получаем финальный результат
    try:
        final_result = self.stage2.output_queue.get(timeout=2.0)
        stage1_result = self.stage1.results[0] if self.stage1.results else None

        print(f"\n[Конвейер] Обработка завершена за {total_time:.6f} секунд")

        return final_result, stage1_result
```

```
            except queue.Empty:
                print("[Конвейер] Не удалось получить результат")
                return None, None

        def run_sequential(self) -> dict:
            """Последовательное выполнение для сравнения"""
            print("\n[Сравнение] Последовательное выполнение...")
            start_time = time.time()

            # Вычисляем сумму ряда
            calculator = SeriesCalculator(self.x, self.epsilon)
            series_result = calculator.calculate_sum()

            # Вычисляем контрольное значение
            y = self.x / 2

            # Вычисляем ошибки
            absolute_error = abs(series_result.value - y)
            relative_error = absolute_error / abs(y) if y != 0 else float("inf")

            end_time = time.time()

            result = {
                "series_value": series_result.value,
                "control_value": y,
                "absolute_error": absolute_error,
                "relative_error": relative_error,
                "iterations": series_result.iterations,
                "total_time": end_time - start_time,
                "is_sequential": True,
            }

            return result

    def analyze_series(x: float, epsilon: float = 1e-7):
        """Анализ ряда и построение графиков"""
        calculator = SeriesCalculator(x, epsilon)

        # Вычисляем частичные суммы для анализа
        partial_sums = []
        errors = []
        n_values = []

        sum_value = 0.0
        n = 1
        term_value = calculator.term(n)
        y = x / 2

        while len(partial_sums) < 30:  # Первые 30 членов для анализа
            sum_value += term_value
            partial_sums.append(sum_value)
            errors.append(abs(sum_value - y))
```

```
        n_values.append(n)

        n += 1
        term_value = calculator.term(n)

    # Строим графики
    fig, axes = plt.subplots(2, 2, figsize=(12, 10))

    # График 1: Частичные суммы
    axes[0, 0].plot(n_values, partial_sums, "b-", linewidth=2, label="S(n)")
    axes[0, 0].axhline(y=y, color="r", linestyle="--", label=f"y = x/2 = {y:.4f}")
    axes[0, 0].set_xlabel("Номер члена ряда (n)")
    axes[0, 0].set_ylabel("Частичная сумма")
    axes[0, 0].set_title("Сходимость ряда к контрольному значению")
    axes[0, 0].legend()
    axes[0, 0].grid(True, alpha=0.3)

    # График 2: Абсолютная ошибка
    axes[0, 1].semilogy(n_values, errors, "g-", linewidth=2, label="Ошибка")
    axes[0, 1].axhline(y=epsilon, color="r", linestyle="--", label=f"ε = {epsilon:.0e}")
    axes[0, 1].set_xlabel("Номер члена ряда (n)")
    axes[0, 1].set_ylabel("Абсолютная ошибка (log scale)")
    axes[0, 1].set_title("Убывание абсолютной ошибки")
    axes[0, 1].legend()
    axes[0, 1].grid(True, alpha=0.3)

    # График 3: Члены ряда
    term_values = [calculator.term(n) for n in n_values]
    axes[1, 0].plot(n_values, term_values, "m-", linewidth=2, label="a_n")
    axes[1, 0].axhline(y=0, color="k", linestyle="-", alpha=0.3)
    axes[1, 0].set_xlabel("Номер члена ряда (n)")
    axes[1, 0].set_ylabel("Значение члена ряда")
    axes[1, 0].set_title("Члены знакочередующегося ряда")
    axes[1, 0].legend()
    axes[1, 0].grid(True, alpha=0.3)

    # График 4: Относительная ошибка
    relative_errors = [e / abs(y) for e in errors]
    axes[1, 1].semilogy(
        n_values, relative_errors, "c-", linewidth=2, label="Отн. ошибка"
    )
    axes[1, 1].set_xlabel("Номер члена ряда (n)")
    axes[1, 1].set_ylabel("Относительная ошибка (log scale)")
    axes[1, 1].set_title("Относительная ошибка")
    axes[1, 1].legend()
    axes[1, 1].grid(True, alpha=0.3)

    plt.suptitle(f"Анализ ряда при x = {x:.4f} (π/2 ≈ 1.5708)")
    plt.tight_layout()
    plt.show()

def main():
```

```
    """Основная функция программы"""
    print("\n" + "=" * 70)
    print("КОНВЕЙЕРНАЯ ОБРАБОТКА С ИСПОЛЬЗОВАНИЕМ
МНОГОПОТОЧНОСТИ")
    print("=" * 70)
    print("Задание:")
    print("  1. Вычислить сумму ряда S = Σ (-1)^(n+1) * sin(nx) / n")
    print("  2. Сравнить результат с контрольной функцией y = x/2")
    print("  3. Организовать конвейер: вычисление -> обработка")
    print("=" * 70)

    # Параметры задачи
    x = -math.pi / 2
    epsilon = 1e-7

    print(f"\nПараметры задачи:")
    print(f"  x = -π/2 = {x:.10f}")
    print(f"  Контрольная функция: y = x/2 = {x/2:.10f}")
    print(f"  Требуемая точность: ε = {epsilon:.2e}")

    # Создаем и запускаем конвейер
    pipeline = Pipeline(x, epsilon)

    # Запускаем конвейерную обработку
    pipeline_result, series_result = pipeline.run()

    # Запускаем последовательное выполнение для сравнения
    sequential_result = pipeline.run_sequential()

    # Выводим результаты
    print("\n" + "=" * 70)
    print("РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЙ")
    print("=" * 70)

    if pipeline_result:
        print("\nКонвейерная обработка:")
        print(f"  Значение ряда S = {pipeline_result['series_value']:.10f}")
        print(f"  Контрольное значение y = {pipeline_result['control_value']:.10f}")
        print(f"  Абсолютная ошибка = {pipeline_result['absolute_error']:.2e}")
        print(f"  Относительная ошибка = {pipeline_result['relative_error']:.2e}")
        print(f"  Итераций: {pipeline_result['iterations']}")
        print(f"  Общее время: {pipeline_result['total_time']:.6f} с")

    if sequential_result:
        print("\nПоследовательное выполнение:")
        print(f"  Значение ряда S = {sequential_result['series_value']:.10f}")
        print(f"  Контрольное значение y = {sequential_result['control_value']:.10f}")
        print(f"  Абсолютная ошибка = {sequential_result['absolute_error']:.2e}")
        print(f"  Относительная ошибка = {sequential_result['relative_error']:.2e}")
        print(f"  Итераций: {sequential_result['iterations']}")
        print(f"  Общее время: {sequential_result['total_time']:.6f} с")
```

```
# Сравнение производительности
if pipeline_result and sequential_result:
    print("\n" + "=" * 70)
    print("СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ")
    print("=" * 70)

    pipeline_time = pipeline_result["total_time"]
    sequential_time = sequential_result["total_time"]

    if pipeline_time > 0 and sequential_time > 0:
        speedup = sequential_time / pipeline_time
        efficiency = (speedup / 2) * 100  # 2 потока в конвейере

        print(f"  Время конвейера: {pipeline_time:.6f} с")
        print(f"  Время последовательного: {sequential_time:.6f} с")
        print(f"  Ускорение: {speedup:.2f}x")
        print(f"  Эффективность (для 2 потоков): {efficiency:.1f}%")

        if speedup > 1:
            print(f"  ✓ Конвейер быстрее на {(speedup - 1) * 100:.1f}%")
        else:
            print(f"  ✗ Последовательное выполнение быстрее")

# Проверка точности
print("\n" + "=" * 70)
print("ПРОВЕРКА ТОЧНОСТИ")
print("=" * 70)

if pipeline_result:
    abs_error = pipeline_result["absolute_error"]
    if abs_error < epsilon:
        print(f"  ✓ Точность достигнута: {abs_error:.2e} < {epsilon:.2e}")
    else:
        print(f"  ✗ Точность не достигнута: {abs_error:.2e} > {epsilon:.2e}")

# Математический анализ
print("\n" + "=" * 70)
print("МАТЕМАТИЧЕСКИЙ АНАЛИЗ РЯДА")
print("=" * 70)

calculator = SeriesCalculator(x, epsilon)
print(f"Для x = -π/2:")
print(f"  sin(x) = sin(-π/2) = {math.sin(x):.6f}")
print(f"  sin(2x) = sin(-π) = {math.sin(2*x):.6f}")
print(f"  sin(3x) = sin(-3π/2) = {math.sin(3*x):.6f}")
print(f"  sin(4x) = sin(-2π) = {math.sin(4*x):.6f}")

print(f"\nПервые члены ряда:")
for n in range(1, 6):
    term = calculator.term(n)
    print(f"  n={n}: a_n = {term:.10f}")
```

```
    # Теоретическое значение
    print(f"\nТеоретическое значение ряда для x = -π/2:")
    print(f"  S = x/2 = {x/2:.10f}")

    # Запускаем анализ сходимости
    print("\n" + "=" * 70)
    print("АНАЛИЗ СХОДИМОСТИ РЯДА")
    print("=" * 70)
    print("Строим графики сходимости...")

    analyze_series(x, epsilon)

    print("\n" + "=" * 70)
    print("ВЫЧИСЛЕНИЯ ЗАВЕРШЕНЫ")
    print("=" * 70)

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("\n\nПрограмма прервана пользователем")
    except Exception as e:
        print(f"\nОшибка: {e}")
```

```
КОНВЕЙЕРНАЯ ОБРАБОТКА С ИСПОЛЬЗОВАНИЕМ МНОГОПОТОЧНОСТИ
======================================================================
Задание:
  1. Вычислить сумму ряда S = Σ (-1)^(n+1) * sin(nx) / n
  2. Сравнить результат с контрольной функцией y = x/2
  3. Организовать конвейер: вычисление -> обработка
======================================================================

Параметры задачи:
  x = -π/2 = -1.5707963268
  Контрольная функция: y = x/2 = -0.7853981634
  Требуемая точность: ε = 1.00e-07

======================================================================
Запуск конвейерной обработки
======================================================================
Параметры:
  x = -1.570796
  Контрольное значение y = x/2 = -0.7853981634
  Точность ε = 1.00e-07

[Конвейер] Запуск стадий...
[Первая функция (ряд)] Вычисление завершено: S = -1.0000000000, итераций: 1, время: 0.000000 с
[Конвейер] Ожидание завершения вычислений...
[Вторая функция (обработка)] Обработка завершена: абс. ошибка = 2.15e-01, отн. ошибка = 2.73e-01

[Конвейер] Обработка завершена за 0.011516 секунд
```

Рисунок 1. Пример работы кода

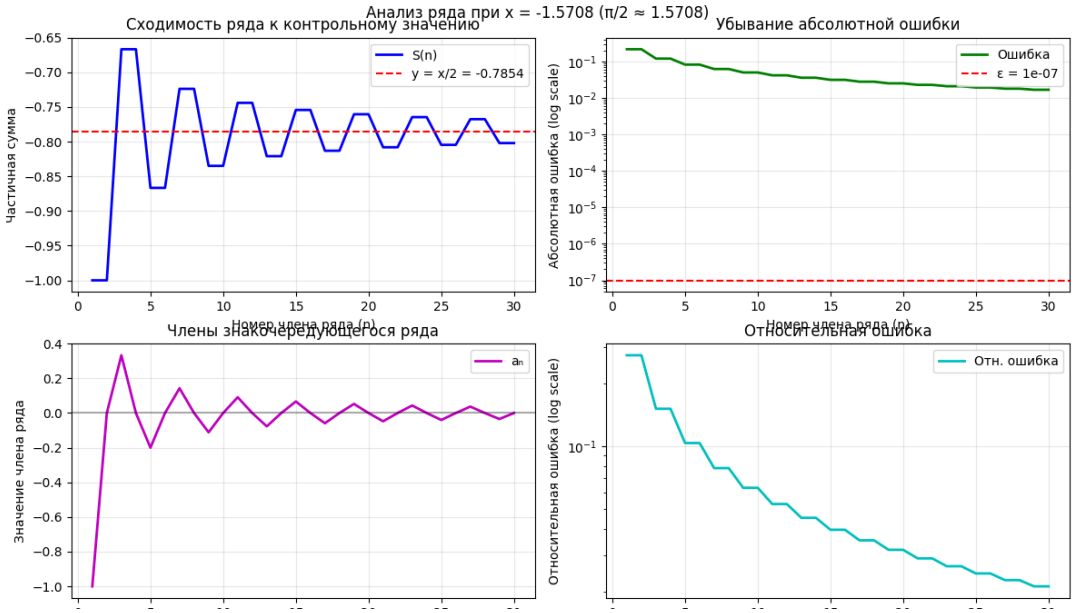

Рисунок 2. Графики примера

Вывод: приобрел навыки использования примитивов синхронизации при написании программ с помощью языка программирования Python версии 3.x.