Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

Лабораторная работа 3
по дисциплине «Современные проблемы информатики»

Выполнил: ст. гр.
ЗМП-41 Лёвкин И. А.

Проверил: Лихачёв А.В.

Новосибирск 2025

Постановка задачи

1. Разработать модель управления доступом нескольких параллельных процессов к одному файлу. Предполагается, что в системе имеются процессы чтения и записи, обращающиеся к рассматриваемому файлу. Одновременно с ним могут работать несколько процессов чтения. Но если работает процесс записи, то никакой другой процесс не может обратиться к файлу, пока запись полностью не завершится. Также считается, что передача управления между процессами происходит случайным образом. Причём вероятность того, что управление будет передано некоторому процессу пропорциональна его приоритету.
2. Построить алгоритм, соответствующий разработанной модели. Функционирование компьютера представляется бесконечным циклом, при входе в очередной период которого происходит передача управления между процессами. За один период может быть считан (записан) один символ из файла. В случае чтения символ помещается в буфер процесса на первое свободное место (заполнение буфера производится последовательно). Когда управление передаётся процессу записи, все процессы чтения прекращаются.
3. Результаты работы представить в виде зависимостей числа считанных символов каждым из процессов чтения от приоритета процесса записи.

Результаты:

```
--- Запуск симуляции с приоритетом писателя 1 ---

Приоритет писателя: 1
Читатель-1 прочитал символов: 31
Читатель-2 прочитал символов: 31
Читатель-3 прочитал символов: 31

--- Запуск симуляции с приоритетом писателя 2 ---

Приоритет писателя: 2
Читатель-1 прочитал символов: 56
Читатель-2 прочитал символов: 56
Читатель-3 прочитал символов: 56

--- Запуск симуляции с приоритетом писателя 3 ---

Приоритет писателя: 3
Читатель-1 прочитал символов: 81
Читатель-2 прочитал символов: 81
Читатель-3 прочитал символов: 81

--- Запуск симуляции с приоритетом писателя 4 ---

Приоритет писателя: 4
Читатель-1 прочитал символов: 104
Читатель-2 прочитал символов: 104
Читатель-3 прочитал символов: 104
```

```
--- Запуск симуляции с приоритетом писателя 5 ---

Приоритет писателя: 5
Читатель-1 прочитал символов: 124
Читатель-2 прочитал символов: 124
Читатель-3 прочитал символов: 124

--- Запуск симуляции с приоритетом писателя 6 ---

Приоритет писателя: 6
Читатель-1 прочитал символов: 144
Читатель-2 прочитал символов: 144
Читатель-3 прочитал символов: 144

--- Запуск симуляции с приоритетом писателя 7 ---

Приоритет писателя: 7
Читатель-1 прочитал символов: 164
Читатель-2 прочитал символов: 164
Читатель-3 прочитал символов: 164

--- Запуск симуляции с приоритетом писателя 8 ---

Приоритет писателя: 8
Читатель-1 прочитал символов: 184
Читатель-2 прочитал символов: 184
Читатель-3 прочитал символов: 184

--- Запуск симуляции с приоритетом писателя 9 ---

Приоритет писателя: 9
Читатель-1 прочитал символов: 204
Читатель-2 прочитал символов: 204
Читатель-3 прочитал символов: 204

--- Запуск симуляции с приоритетом писателя 10 ---

Приоритет писателя: 10
Читатель-1 прочитал символов: 224
Читатель-2 прочитал символов: 224
Читатель-3 прочитал символов: 224
```

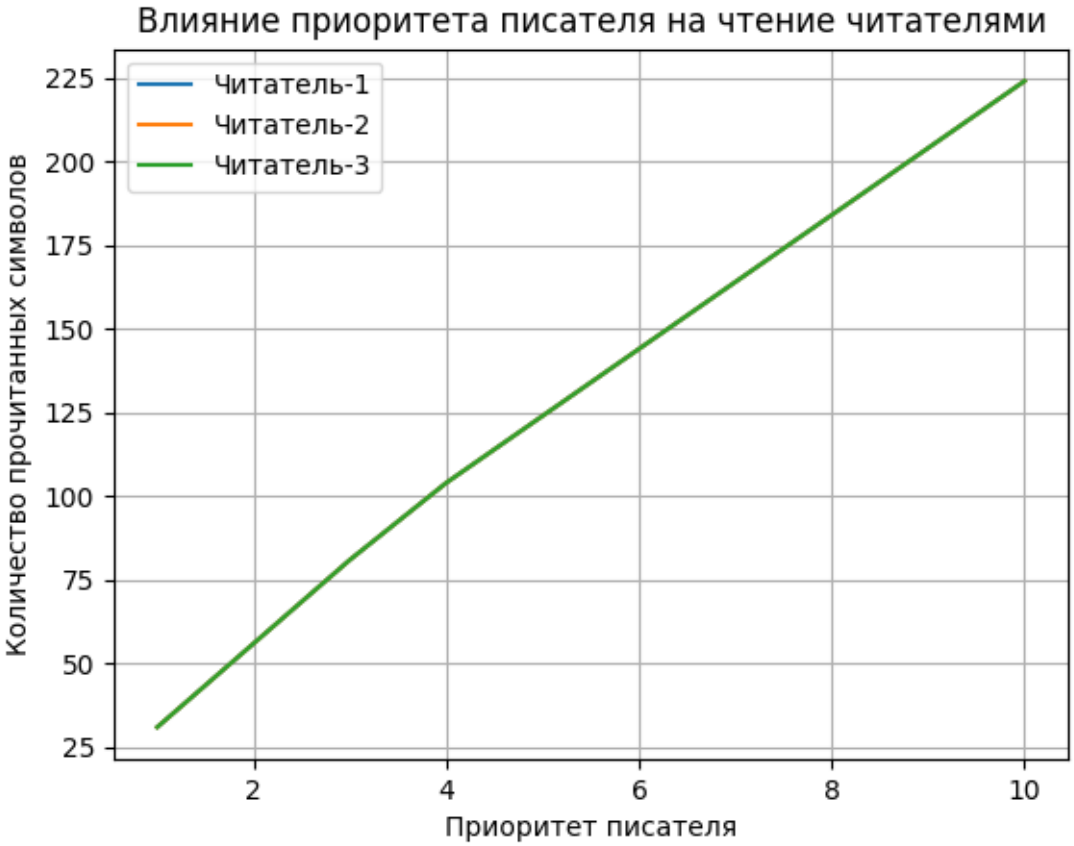

Рис. 1. Зависимость числа считанных символов от приоритета писателя

Листинг

```
import threading
import random
import time
import matplotlib.pyplot as plt

file_path = "testfile.txt"
buffer_size = 1000

# Глобальные переменные для синхронизации
file_lock = threading.Lock()
read_count = 0
read_count_lock = threading.Lock()
write_lock = threading.Lock()


class Reader(threading.Thread):
    def __init__(self, name, priority, file_path):
        super().__init__()
        self.name = name
        self.priority = priority
        self.file_path = file_path
        self.buffer = []
        self.position = 0  # позиция чтения в файле
        self.read_chars = 0

    def run(self):
        global read_count
        while True:
```

```
            # Проверим конец файла
            with file_lock:
                try:
                    with open(self.file_path, "r", encoding="utf-8") as
f:
                        f.seek(0, 2)
                        file_len = f.tell()
                except Exception as e:
                    print(f"Ошибка открытия файла: {e}")
                    break

            if self.position >= file_len or len(self.buffer) >=
buffer_size:
                break

            # Захватываем право читать
            with read_count_lock:
                if read_count == 0:
                    write_lock.acquire()
                read_count += 1

            with file_lock:
                with open(self.file_path, "r", encoding="utf-8") as f:
                    f.seek(self.position)
                    c = f.read(1)
                    if c:
                        self.buffer.append(c)
                        self.position += len(c.encode("utf-8"))  #
Смещение в байтах
                        self.read_chars += 1
```

```
                    # print(f"{self.name} прочитал символ: '{c}'")

            with read_count_lock:
                read_count -= 1
                if read_count == 0:
                    write_lock.release()

            time.sleep(0.01)  # Имитация времени чтения


class Writer(threading.Thread):
    def __init__(self, name, priority, file_path, max_writes=20):
        super().__init__()
        self.name = name
        self.priority = priority
        self.file_path = file_path
        self.max_writes = max_writes
        self.writes_done = 0

    def run(self):
        while self.writes_done < self.max_writes:
            write_lock.acquire()
            with file_lock:
                try:
                    with open(self.file_path, "a", encoding="utf-8") as
f:
                        c =
random.choice("абвгдеёжзиклмнопрстуфхцчшщъыьэюя ")
                        f.write(c)
                        self.writes_done += 1
```

```
                        # print(f"{self.name} записал символ: '{c}'")
                except Exception as e:
                    print(f"Ошибка записи в файл: {e}")
            write_lock.release()
            time.sleep(0.05)


def simulate(priorities_writer):
    readers = [
        Reader(f"Читатель-{i+1}", priority=1, file_path=file_path) for i
in range(3)
    ]
    writer = Writer("Писатель", priority=priorities_writer,
file_path=file_path)

    # Запускаем потоки
    for r in readers:
        r.start()
    writer.start()

    # Моделируем вытесняющую многозадачность с выбором по приоритетам
    processes = readers + [writer]

    while any(p.is_alive() for p in processes):
        alive = [p for p in processes if p.is_alive()]
        weights = [p.priority for p in alive]
        chosen = random.choices(alive, weights=weights, k=1)[0]
        # Квант времени для выбранного процесса — имитация через sleep
        time.sleep(0.02)
```

```
    # Собираем статистику по чтению
    read_counts = [r.read_chars for r in readers]
    print(f"\nПриоритет писателя: {priorities_writer}")
    for i, r in enumerate(readers):
        print(f"{r.name} прочитал символов: {read_counts[i]}")
    return read_counts


def main():
    # Создаем или очищаем файл перед тестом
    with open(file_path, "w", encoding="utf-8") as f:
        f.write("Начальный текст файла.\n")

    priority_range = range(1, 11)  # Приоритеты писателя от 1 до 10
    results = []

    for pr in priority_range:
        # Для каждого приоритета писателя запускаем отдельную симуляцию
        print(f"\n--- Запуск симуляции с приоритетом писателя {pr} ---")
        read_counts = simulate(pr)
        results.append(read_counts)

    # Строим график
    import numpy as np

    results = np.array(results)
    for i in range(results.shape[1]):
        plt.plot(priority_range, results[:, i], label=f"Читатель-{i+1}")
```

```
    plt.xlabel("Приоритет писателя")
    plt.ylabel("Количество прочитанных символов")
    plt.title("Влияние приоритета писателя на чтение читателями")
    plt.legend()
    plt.grid(True)
    plt.show()


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