**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

**Тема: «Межпроцессное взаимодействие»**

Студента гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Половникова А.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимофеев А.В.

Санкт-Петербург

2025

**Цель работы.**

исследовать инструменты и механизмы взаимодействия процессов в Windows.

**Задание 4.1.**

Реализация решения задачи о читателях-писателях.

**Указания к выполнению.**

1. Выполнить решение задачи о читателях-писателях, для чего необходимо разработать консольные приложения «Читатель» и «Писатель»: одновременно запущенные экземпляры процессов-читателей и процессов писателей должны совместно работать с буферной памятью в виде проецируемого файла:

o размер страницы буферной памяти равен размеру физической страницы оперативной памяти;

o число страниц буферной памяти равно сумме цифр в номере студенческого билета без учета первой цифры.

страницы буферной памяти должны быть заблокированы в оперативной памяти (функция VirtualLock);

длительность выполнения процессами операций «чтения» и «записи» задается случайным образом в диапазоне от 0,5 до 1,5 сек.;

для синхронизации работы процессов необходимо использовать объекты синхронизации типа «семафор» или «мьютекс»;

процессы-читатели и процессы-писатели ведут свои журнальные файлы, в которые регистрируют переходы из одного «состояния» в другое (начало ожидания, запись или чтение, переход к освобождению) с указанием кода времени (функция TimeGetTime). Для состояний «запись» и «чтение» необходимо также запротоколировать номер рабочей страницы.

2. Запустите приложения читателей и писателей, суммарное количество одновременно работающих читателей и писателей должно быть не менее числа страниц буферной памяти. Проверьте функционирование приложений, проанализируйте журнальные файлы процессов, постройте сводные графики смены «состояний» для не менее 5 процессов-читателей и 5 процессов-писателей, дайте свои комментарии относительно переходов процессов из одного состояния в другое.

3. Постройте графики в Excel/Python и дайте свои комментарии:

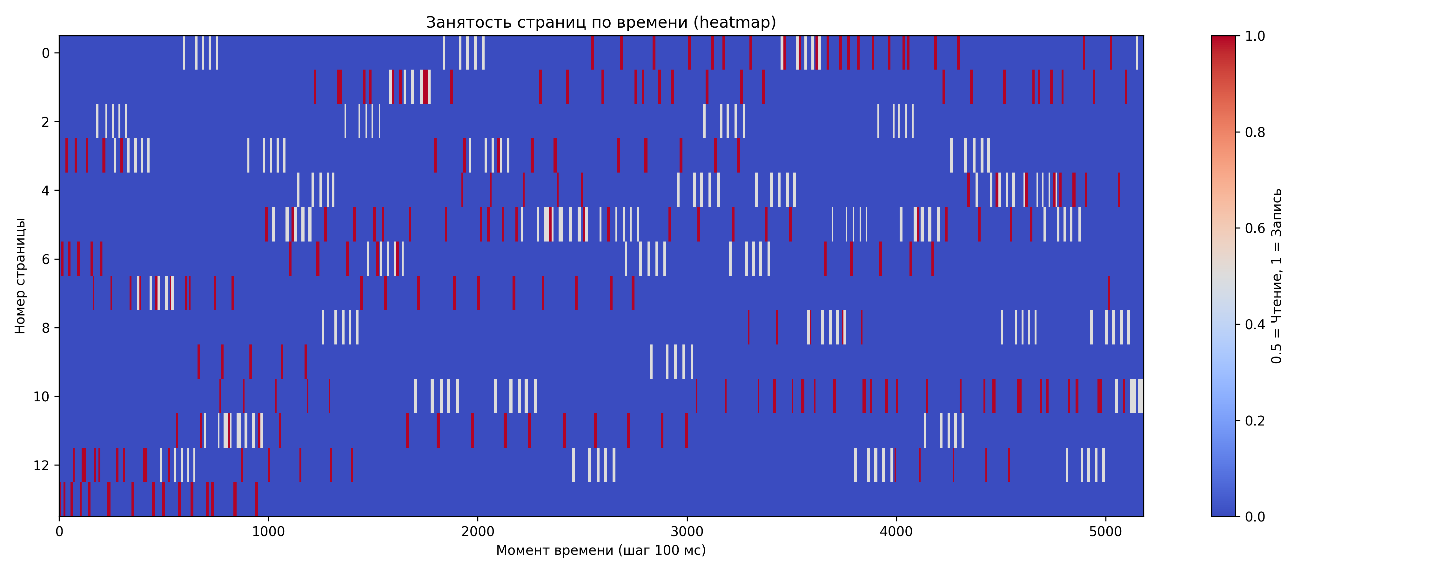
 Смена состояний процессов (пример: читатель 1 → ожидание → чтение

→ освобождение).

 Занятость страниц во времени (визуализация через heatmap).

4. Подготовьте итоговый отчет с развернутыми выводами по

заданию.



**График 1: Занятость страниц по времени (heatmap)**

**Ось Y (Номер страницы):** Показывает номер страницы памяти, которая используется.

**Ось X (Момент времени):** Представлен в шагах по 100 мс — фактически шкала времени.

**Цветовая шкала:**

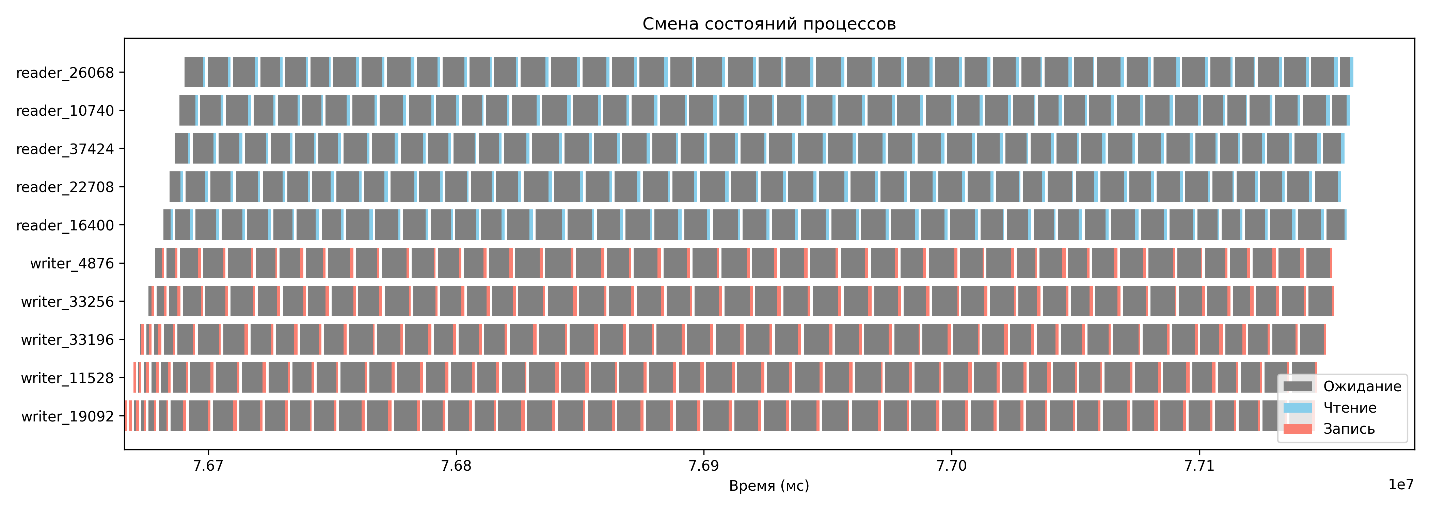
* **0.5 (белый)** = чтение
* **1 (красный)** = запись
* **0 (синий фон)** = страница не используется

**На графике:**

* Активность разбросана по разным страницам и моментам времени.
* Есть страницы с частыми обращениями (например, страница 0, 2, 6 и 12), что может говорить о «горячих страницах».
* Периоды записи (красный) часто чередуются с чтением (белый), это указывает на интенсивную работу с памятью.

**Выводы:**

Механизм кэширования или буферизации работает активно.



**График 2: Смена состояний процессов**

**Ось Y:** Имена процессов (разделены на reader\_... и writer\_...)

**Ось X:** Время в миллисекундах

**Цветовая кодировка:**

* **Серый:** ожидание
* **Голубой:** чтение
* **Красный:** запись

**Что видно:**

* Читатели работают с интервалами — читают, потом ждут, потом снова читают.
* Писатели делают много коротких записей, особенно в нижней части (например, writer\_19092) — возможно, конкурируют за доступ к ресурсу.
* Все процессы проводят значительное время в состоянии ожидания (серый), что может указывать на блокировки или синхронизацию.

**Выводы:**

* Система хорошо распределяет доступ к ресурсу.
* Баланс между читателями и писателями наблюдается, но писатели имеют тенденцию к более частым обращениям.

**Общие наблюдения:**

* **Взаимосвязь между графиками:** Частые записи и чтения на первом графике (heatmap) коррелируют с активностью на втором графике — особенно видно, как писатели делают много коротких операций.

**Вывод**

**Архитектура решения**

Программа реализована в виде двух независимых процессов:

* **Writer** — записывает данные в случайно выбранную страницу общей памяти.
* **Reader** — считывает данные с случайной страницы.

Вся коммуникация происходит через следующие механизмы:

| **Компонент** | **Используемая технология** | **Назначение** |
| --- | --- | --- |
| Разделяемая память | CreateFileMappingW / MapViewOfFile | Хранение общих данных (страницы) |
| Синхронизация | CreateMutexW / OpenMutexW | Исключение конкурентного доступа |
| Логгирование | Файлы .log | Отслеживание временных меток и операций |

**Детали реализации**

**Общие определения (Header.h)**

#define MUTEX\_NAME L"Global\\MyMutex"

#define FILE\_MAPPING\_NAME L"Global\\MySharedMemory"

#define PAGE\_COUNT 16

#define PAGE\_SIZE 4096

void LogToFile(const std::wstring& filename, const std::wstring& message);

**Writer (Процесс записи)**

Writer создаёт:

* **mutex** — для защиты доступа.
* **разделяемую память** — в объёме PAGE\_COUNT \* PAGE\_SIZE.

**Цикл работы:**

1. Логирует начало ожидания ([WAITING]).
2. Захватывает мьютекс (WaitForSingleObject).
3. Выбирает случайную страницу (rand() % PAGE\_COUNT).
4. Записывает в неё строку "PID <ид процесса>".
5. Логирует запись ([WRITING]) и отпускание мьютекса ([RELEASE]).
6. Засыпает на 0.5–1.5 сек, затем ждёт 1 секунду и повторяет.

**Reader (Процесс чтения)**

Reader подключается к:

* **уже существующему мьютексу**
* **существующей разделяемой памяти**, созданной Writer'ом

**Цикл работы:**

1. Логирует ожидание ([WAITING]).
2. Захватывает мьютекс.
3. Выбирает случайную страницу.
4. Считывает данные (strnlen\_s).
5. Выводит результат в консоль.
6. Логирует чтение ([READING]) и отпускание ([RELEASE]).
7. Засыпает так же, как Writer.

**Визуализация (heatmap & timeline)**

**Heatmap**

Отражает распределение доступа по страницам.

**Timeline**

Позволяет выявить взаимные блокировки, конкуренцию, время ожидания.

**Синхронизация**

Оба типа процессов используют один и тот же **mutex** для защиты доступа к разделяемой памяти. Это:

* Исключает гонки и повреждение данных.
* Но блокирует *всех* читателей, если writer внутри критической секции, и наоборот.

Возможны улучшения с помощью **reader-writer lock**.

**Логгирование**

Каждый процесс ведёт лог-файл:

* reader\_<pid>.log
* writer\_<pid>.log

Позволяет:

* Трассировать поведение.
* Восстановить действия по времени.
* Построить визуализации.

**Результаты**

1. Реализована корректная синхронизация доступа к общей памяти между несколькими процессами.
2. Проведена отладка поведения в конкурентной среде.
3. Построены графики, отражающие:
   * Частоту обращений к страницам.
   * Активность и блокировки во времени.

Текст программы

Reader.cpp

#include "Header.h"

#include <iostream>

#include <fstream>

#include <random>

#include <thread>

int main()

{

    HANDLE hMutex = OpenMutexW(MUTEX\_ALL\_ACCESS, FALSE, MUTEX\_NAME);

    if (!hMutex){std::wcerr << L"Error OpenMutex: " << GetLastError() << std::endl;return 1;}

    HANDLE hMapFile = OpenFileMappingW(FILE\_MAP\_READ, FALSE, FILE\_MAPPING\_NAME);

    if (!hMapFile){std::wcerr << L"Error OpenFileMapping: " << GetLastError() << std::endl;return 1;}

    LPVOID pBase = MapViewOfFile(hMapFile, FILE\_MAP\_READ, 0, 0, 0);

    if (!pBase){std::wcerr << L"Error MapViewOfFile: " << GetLastError() << std::endl;return 1;}

    VirtualLock(pBase, PAGE\_COUNT \* PAGE\_SIZE);

    std::wstring logFile = L"reader\_" + std::to\_wstring(GetCurrentProcessId()) + L".log";

    while (true){

        DWORD tickStart = timeGetTime();

        LogToFile(logFile, L"[WAITING] " + std::to\_wstring(tickStart));

        WaitForSingleObject(hMutex, INFINITE);

        int page = rand() % PAGE\_COUNT;

        char\* pageAddr = (char\*)pBase + (page \* PAGE\_SIZE);

        DWORD tickRead = timeGetTime();

        LogToFile(logFile, L"[READING] Page " + std::to\_wstring(page) + L" at " + std::to\_wstring(tickRead));

        std::string readData(pageAddr, pageAddr + strnlen\_s(pageAddr, PAGE\_SIZE));

        std::wcout << L"[" << GetCurrentProcessId() << L"] Прочитал с страницы " << page << L": " << std::wstring(readData.begin(), readData.end()) << std::endl;

        std::this\_thread::sleep\_for(std::chrono::milliseconds(500 + rand() % 1000));

        DWORD tickRelease = timeGetTime();

        LogToFile(logFile, L"[RELEASE] " + std::to\_wstring(tickRelease));

        ReleaseMutex(hMutex);

        std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

    }

    return 0;

}

Writer.cpp

#include "Header.h"

#include <iostream>

#include <fstream>

#include <random>

#include <thread>

int main()

{

    HANDLE hMutex = CreateMutexW(nullptr, FALSE, MUTEX\_NAME);

    if (!hMutex){std::wcerr << L"Error CreateMutex: " << GetLastError() << std::endl; return 1;}

    HANDLE hMapFile = CreateFileMappingW(INVALID\_HANDLE\_VALUE, nullptr, PAGE\_READWRITE, 0, PAGE\_COUNT \* PAGE\_SIZE, FILE\_MAPPING\_NAME);

    if (!hMapFile){std::wcerr << L"Error CreateFileMapping: " << GetLastError() << std::endl; return 1;}

    LPVOID pBase = MapViewOfFile(hMapFile, FILE\_MAP\_WRITE, 0, 0, 0);

    if (!pBase){ std::wcerr << L"Error MapViewOfFile: " << GetLastError() << std::endl; return 1;}

    VirtualLock(pBase, PAGE\_COUNT \* PAGE\_SIZE);

    std::wstring logFile = L"writer\_" + std::to\_wstring(GetCurrentProcessId()) + L".log";

    while (true){

        DWORD tickStart = timeGetTime();

        LogToFile(logFile, L"[WAITING] " + std::to\_wstring(tickStart));

        WaitForSingleObject(hMutex, INFINITE);

        int page = rand() % PAGE\_COUNT;

        char\* pageAddr = (char\*)pBase + (page \* PAGE\_SIZE);

        DWORD tickWrite = timeGetTime();

        LogToFile(logFile, L"[WRITING] Page " + std::to\_wstring(page) + L" at " + std::to\_wstring(tickWrite));

        std::string data = "PID " + std::to\_string(GetCurrentProcessId());

        memcpy(pageAddr, data.c\_str(), min(data.size(), PAGE\_SIZE - 1));

        std::this\_thread::sleep\_for(std::chrono::milliseconds(500 + rand() % 1000));

        DWORD tickRelease = timeGetTime();

        LogToFile(logFile, L"[RELEASE] " + std::to\_wstring(tickRelease));

        ReleaseMutex(hMutex);

        std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

    }

    return 0;

}

Header.h

#pragma once

#include <windows.h>

#include <string>

const wchar\_t\* FILE\_MAPPING\_NAME = L"Global\\MySharedMemory";

const wchar\_t\* MUTEX\_NAME = L"Global\\PageMutex";

const size\_t PAGE\_SIZE = 4096;

const int PAGE\_COUNT = 14;

inline void LogToFile(const std::wstring& filename, const std::wstring& message)

{

FILE\* f;

\_wfopen\_s(&f, filename.c\_str(), L"a+");

if (f)

{

fwprintf(f, L"%s\n", message.c\_str());

fclose(f);

}

}